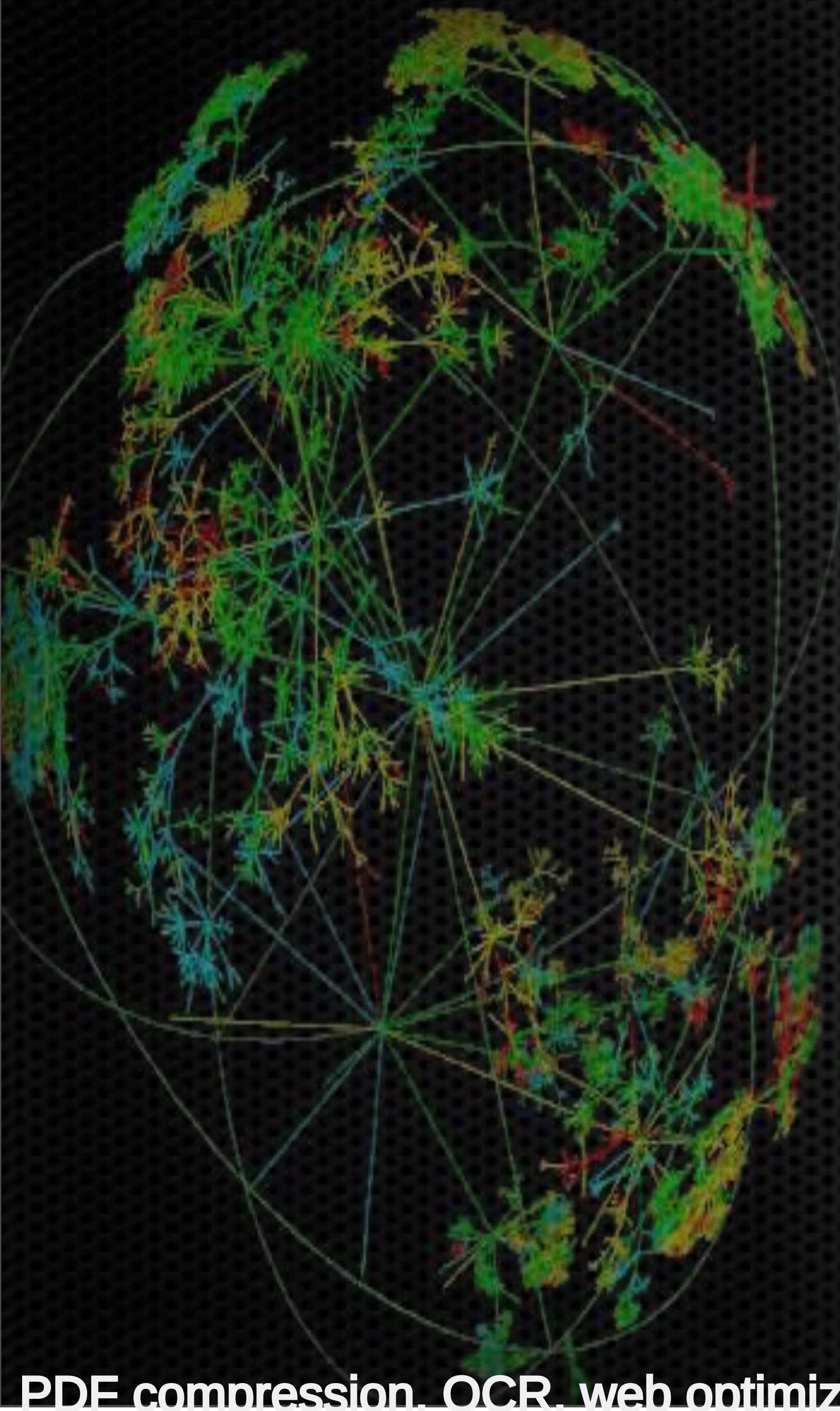
VORTECH BV
THE SCIENTIFIC SOFTWARE ENGINEERS

"LINC"
learning about interacting networks in climate



“computational issues” Complex Network Analysis

Software Engineering Case study: “LINC software for complex network analysis”

H. Ihshaish & J. **Dijkzeul** VORtech BV.
Delft/NL

LINC School II, Utrecht, The Netherlands

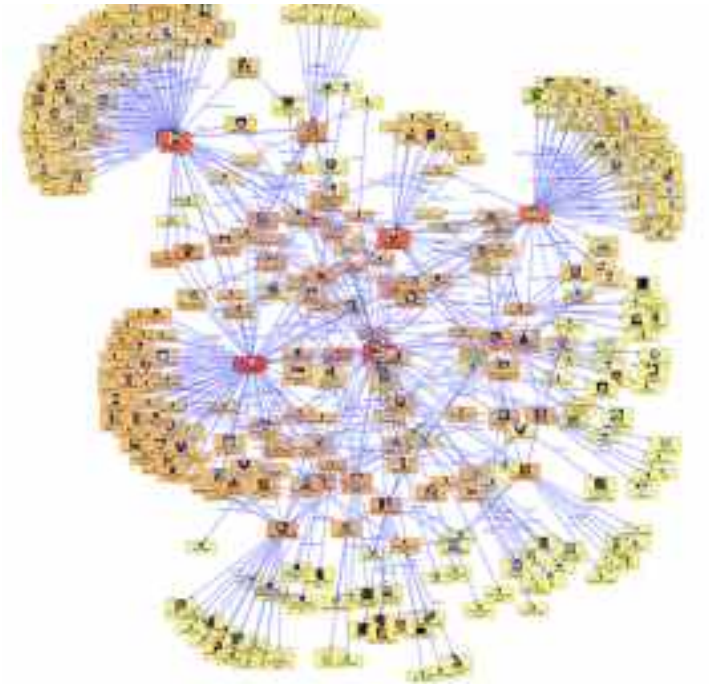
Outline

- Software Engineering
  - ➡ Importance
  - ➡ Process
- Network analysis
  - ➡ Data mining and Networks approach
  - ➡ Computational issues
  - ➡ Parallel network algorithms
- Software Production: parallel “Igraph” algorithms
  - ➡ How is it being developed: “SE”
  - ➡ What next?

Large-scale data analysis

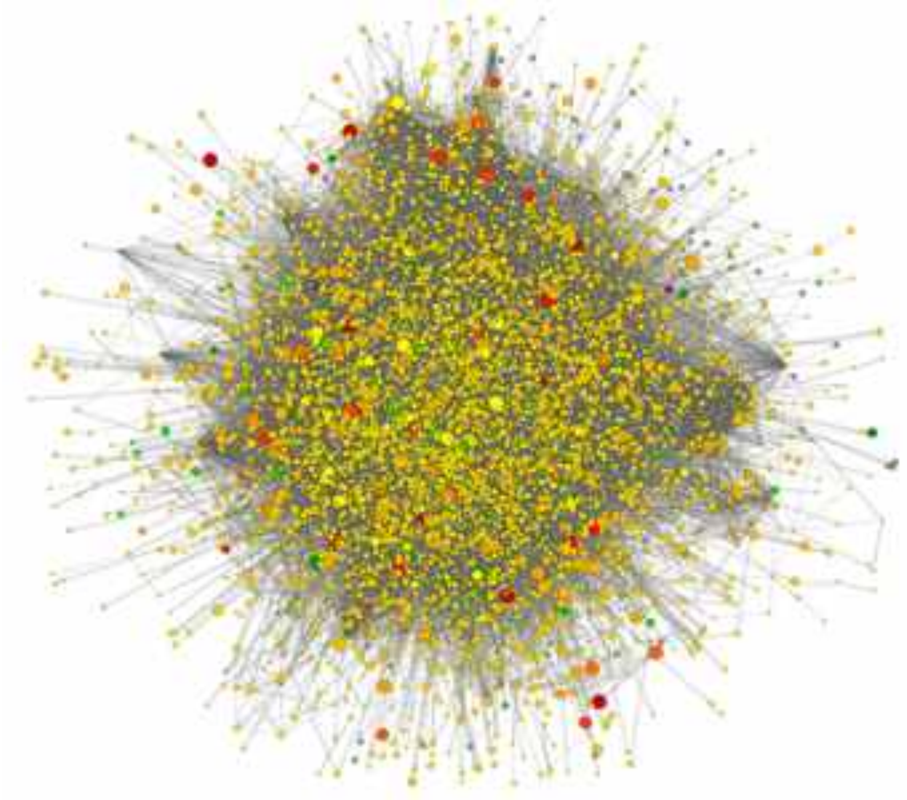
Graph abstractions are very useful to analyze complex data sets.

Social Networks



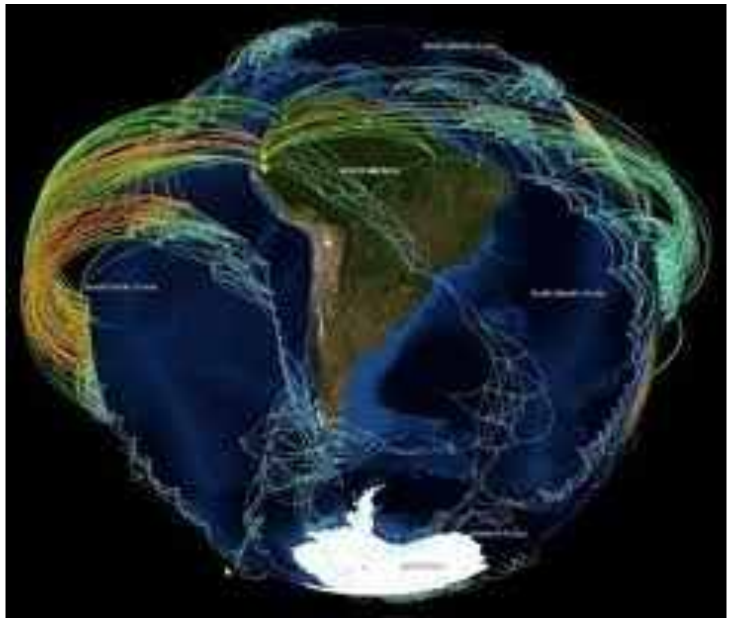
Application: e.g., identifying communities, information spread modeling

BioInformatics



Application: e.g., identifying drug target proteins

Climate??



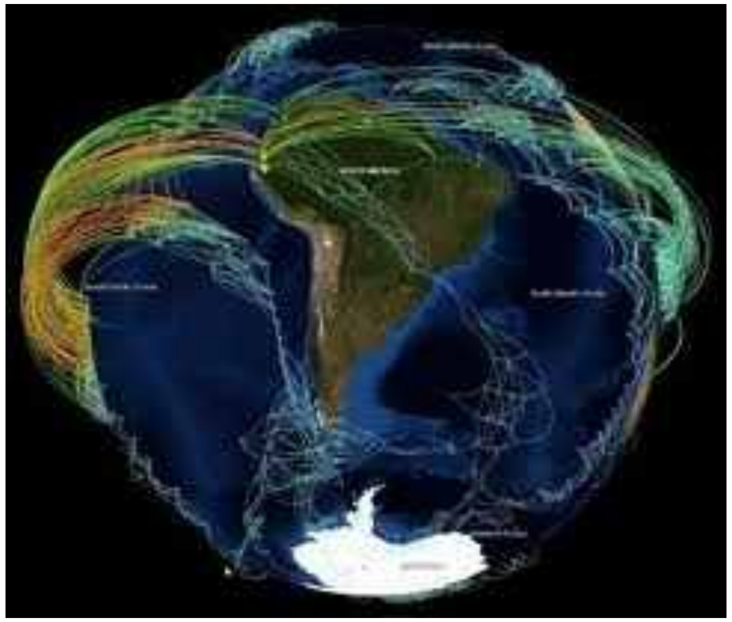
Application: e.g., identify patterns? analyze spatio-temporal interaction of climate data

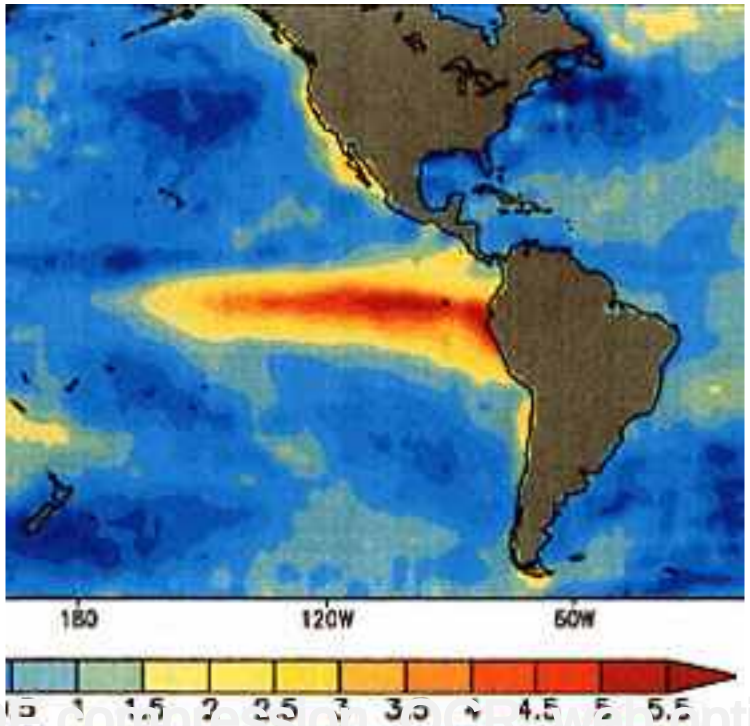
....

- **Sources of data:** simulations, experimental devices, the Internet, sensor networks
- **Challenges:** data size, heterogeneity, uncertainty, data quality

Large-scale data analysis

Graph abstractions are very useful to analyze complex data sets.





Towards better understanding of our climate system

"Complex network theory provides a powerful framework to statistically investigate the topology of local and non-local statistical interrelationships, i.e. teleconnections, in the climate system. Climate networks constructed from the same global climatological data set using the linear Pearson correlation coefficient or the nonlinear mutual information as a measure of dynamical similarity between regions, are compared systematically on local, mesoscopic and global topological scales."

J.F. Donges, Y. Zou, N. Marwan and J. Kurths, "Complex Networks in Climate Dynamics", *Eur. Phys. J. Special Topics 174, 157–179 (2009).*

Computer science,
Scientific and Combinatorial Scientific Computing,
High Performance Computing

GUEST EDITORS' INTRODUCTION

Combinatorics in Computing

Combinatorial questions and problems have been important to computational modeling since the earliest days of digital machines, but until recently, the computational science community has considered these topics to be a necessary substructure, not a main subject of interest. Here's an example of how things used to be: once upon a time, one of us worked as a beginning programmer for an entity called "The Westinghouse Astronuclear Lab." Much time and mental effort was spent writing a code that read from tape a matrix *A*, did a few computations to reorder *A*'s entries and make it *A*1, and then transposed *A*1 and wrote the transpose onto tape. The matrices, of course, didn't fit the core—in fact, the computer couldn't even store one row. We had to invent various "tricks" for marking where the data came from and where it was to go. Interestingly, many of the engineers working on the problem had trouble understanding why reorganizing the data should be slightly hard, but those who'd written programs for the IBM 7090 got the point right away.

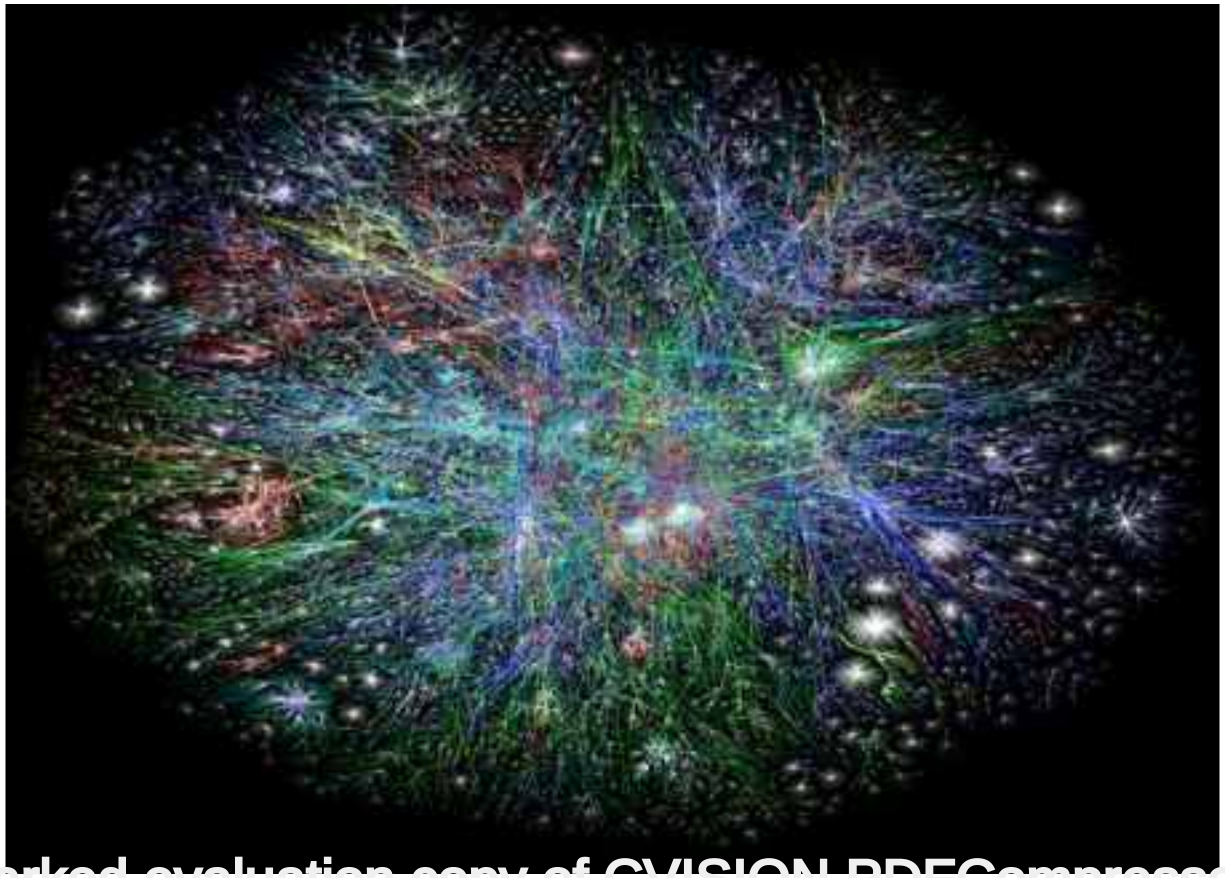
In reality, of course, questions about how to arrange data to minimize computation and provide rapid access have always been crucial to determining the success or failure of computational projects and computing machines. In a 1970 paper appropriately entitled "von Neumann's First Computer Program" (*ACM Computing Surveys*, vol. 2, no. 4, 1970, pp. 247–260), Donald Knuth discussed what probably was the first program John von Neumann wrote for a stored program machine—a program for sorting data into non-decreasing order. According to Knuth, von Neumann chose this example because

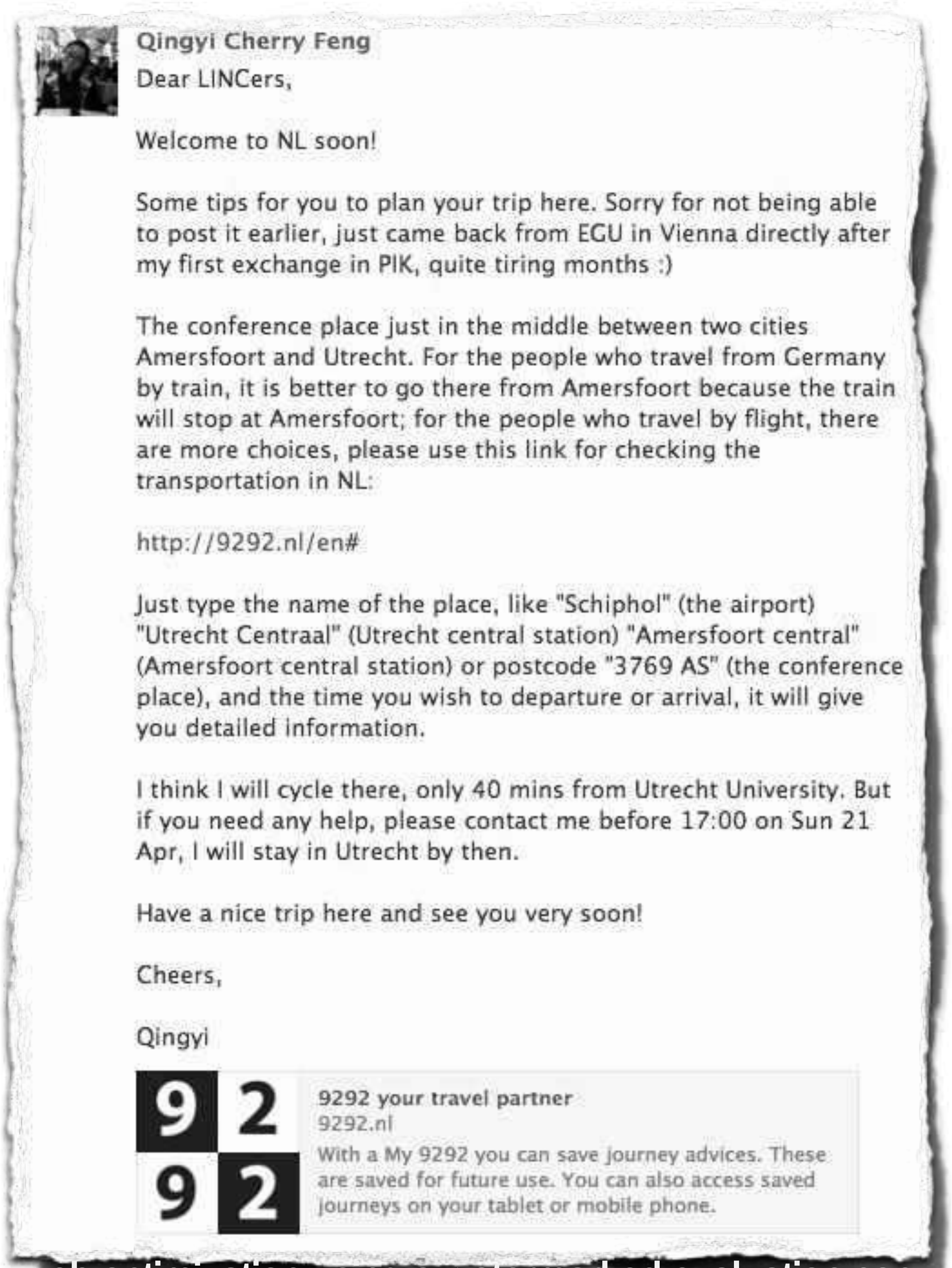
1521-9615/08/$25.00 © 2008 IEEE
Copublished by the IEEE CS and the AIP

Isabel Beichl
National Institute of Standards and Technology
Francis Sullivan
IDA Center for Computing Sciences

"Combinatorial problems generated by challenges in data mining and related topics are now central to computational science. Finally, there's the Internet itself, probably the largest graph-theory problem ever confronted."

- Isabel Beichl & Francis Sullivan, 2008





Qingyi Cherry Feng

Dear LINCers,

Welcome to NL soon!

Some tips for you to plan your trip here. Sorry for not being able to post it earlier, just came back from EGU in Vienna directly after my first exchange in PIK, quite tiring months :)

The conference place just in the middle between two cities Amersfoort and Utrecht. For the people who travel from Germany by train, it is better to go there from Amersfoort because the train will stop at Amersfoort; for the people who travel by flight, there are more choices, please use this link for checking the transportation in NL:

http://9292.nl/en#

Just type the name of the place, like "Schiphol" (the airport) "Utrecht Centraal" (Utrecht central station) "Amersfoort central" (Amersfoort central station) or postcode "3769 AS" (the conference place), and the time you wish to departure or arrival, it will give you detailed information.

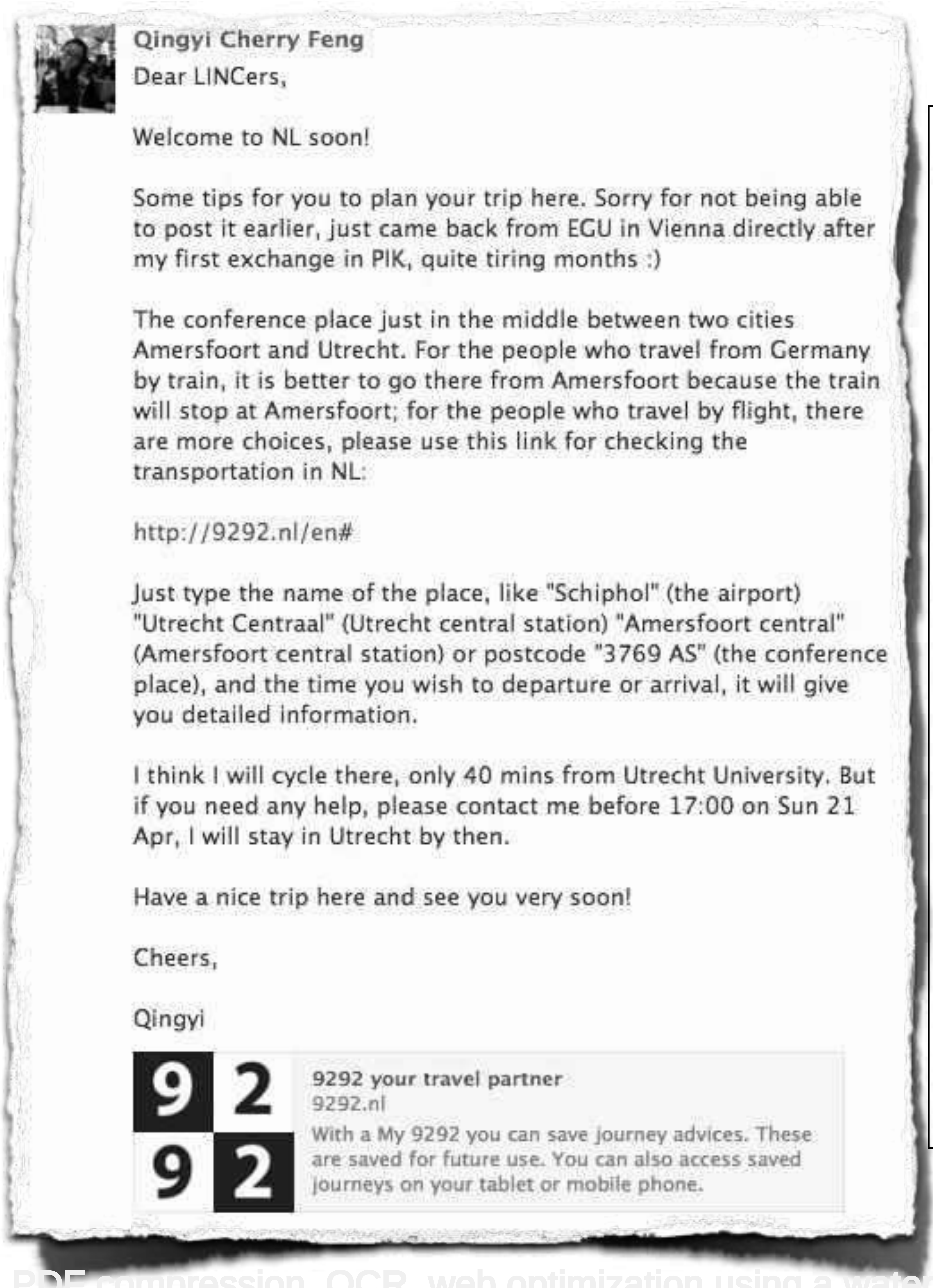
I think I will cycle there, only 40 mins from Utrecht University. But if you need any help, please contact me before 17:00 on Sun 21 Apr, I will stay in Utrecht by then.

Have a nice trip here and see you very soon!

Cheers,

Qingyi

9292 your travel partner
9292.nl
With a My 9292 you can save journey advices. These are saved for future use. You can also access saved journeys on your tablet or mobile phone.



Qingyi Cherry Feng

Dear LINCers,

Welcome to NL soon!

Some tips for you to plan your trip here. Sorry for not being able to post it earlier, just came back from EGU in Vienna directly after my first exchange in PIK, quite tiring months :)

The conference place just in the middle between two cities Amersfoort and Utrecht. For the people who travel from Germany by train, it is better to go there from Amersfoort because the train will stop at Amersfoort; for the people who travel by flight, there are more choices, please use this link for checking the transportation in NL:

http://9292.nl/en#

Just type the name of the place, like "Schiphol" (the airport) "Utrecht Centraal" (Utrecht central station) "Amersfoort central" (Amersfoort central station) or postcode "3769 AS" (the conference place), and the time you wish to departure or arrival, it will give you detailed information.

I think I will cycle there, only 40 mins from Utrecht University. But if you need any help, please contact me before 17:00 on Sun 21 Apr, I will stay in Utrecht by then.

Have a nice trip here and see you very soon!

Cheers,

Qingyi

9292 your travel partner
9292.nl
With a My 9292 you can save journey advices. These are saved for future use. You can also access saved journeys on your tablet or mobile phone.

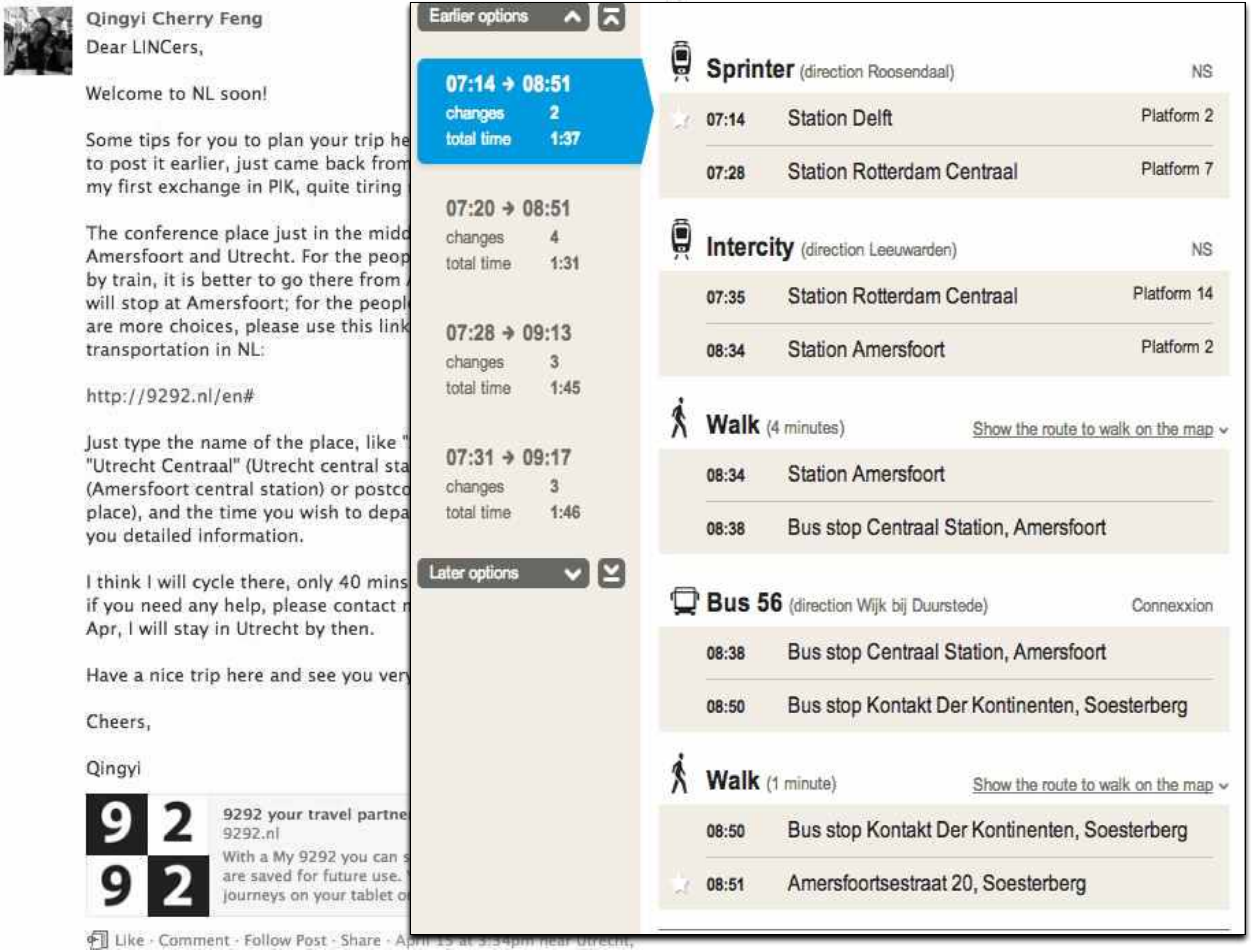
Combinatorial problems arise in many applications:

- find shortest/cheapest round trips (TSP)
- planning, scheduling, etc
- internet data packet routing
- protein structure prediction

Combinatorial problems involve finding a grouping and ordering of a discrete finite set of objects, satisfying a given set of conditions..

Candidate solutions: solution components that might be encountered during a solutions attempt, but need not satisfy given conditions..

Solutions are candidate solutions that satisfy all given conditions...



Qingyi Cherry Feng
Dear LINCers,

Welcome to NL soon!

Some tips for you to plan your trip he
to post it earlier, just came back from
my first exchange in PIK, quite tiring

The conference place just in the midd
Amersfoort and Utrecht. For the peop
by train, it is better to go there from
will stop at Amersfoort; for the peopl
are more choices, please use this link
transportation in NL:

http://9292.nl/en#

Just type the name of the place, like "
"Utrecht Centraal" (Utrecht central sta
(Amersfoort central station) or postco
place), and the time you wish to depa
you detailed information.

I think I will cycle there, only 40 mins
if you need any help, please contact n
Apr, I will stay in Utrecht by then.

Have a nice trip here and see you ver

Cheers,

Qingyi

9292 your travel partne
9292.nl
With a My 9292 you can s
are saved for future use.
journeys on your tablet o

Like · Comment · Follow Post · Share · April 15 at 3:34pm near Utrecht,

07:14 → 08:51
changes 2
total time 1:37

07:20 → 08:51
changes 4
total time 1:31

07:28 → 09:13
changes 3
total time 1:45

07:31 → 09:17
changes 3
total time 1:46

Sprinter (direction Roosendaal) NS

07:14	Station Delft	Platform 2
07:28	Station Rotterdam Centraal	Platform 7

Intercity (direction Leeuwarden) NS

07:35	Station Rotterdam Centraal	Platform 14
08:34	Station Amersfoort	Platform 2

Walk (4 minutes) Show the route to walk on the map

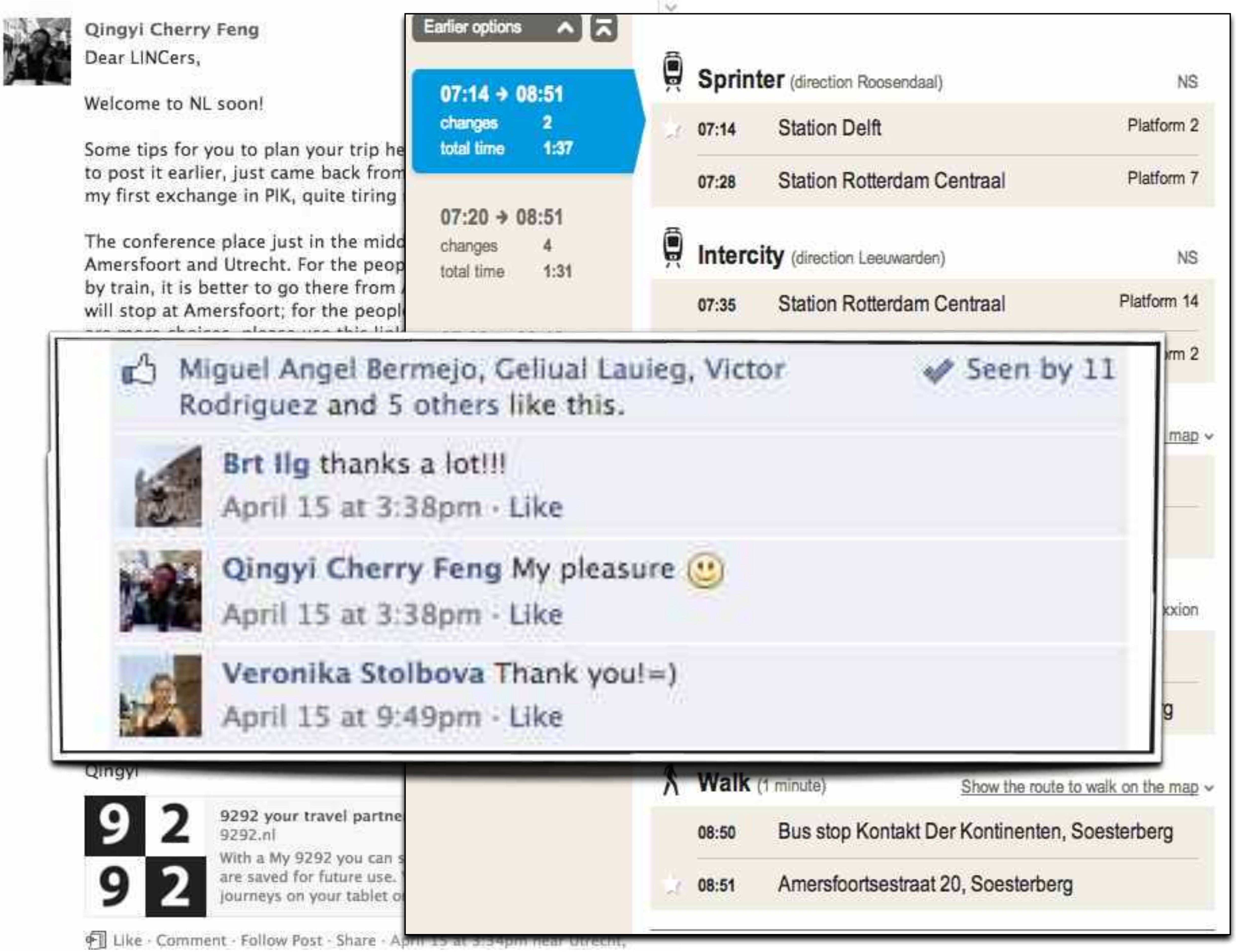
08:34	Station Amersfoort
08:38	Bus stop Centraal Station, Amersfoort

Bus 56 (direction Wijk bij Duurstede) Connexxion

08:38	Bus stop Centraal Station, Amersfoort
08:50	Bus stop Kontakt Der Kontinenten, Soesterberg

Walk (1 minute) Show the route to walk on the map

08:50	Bus stop Kontakt Der Kontinenten, Soesterberg
08:51	Amersfoortsestraat 20, Soesterberg



Qingyi Cherry Feng
Dear LINCers,

Welcome to NL soon!

Some tips for you to plan your trip he
to post it earlier, just came back from
my first exchange in PIK, quite tiring

The conference place just in the midd
Amersfoort and Utrecht. For the peop
by train, it is better to go there from
will stop at Amersfoort; for the peopl

Qingyi

9292 your travel partne
9292.nl
With a My 9292 you can s
are saved for future use.
journeys on your tablet o

Like · Comment · Follow Post · Share · April 15 at 3:34pm near Utrecht,

Earlier options

07:14 → 08:51
changes 2
total time 1:37

07:20 → 08:51
changes 4
total time 1:31

Sprinter (direction Roosendaal) NS

07:14	Station Delft	Platform 2
07:28	Station Rotterdam Centraal	Platform 7

Intercity (direction Leeuwarden) NS

07:35	Station Rotterdam Centraal	Platform 14

Walk (1 minute) Show the route to walk on the map

08:50	Bus stop Kontakt Der Kontinenten, Soesterberg
08:51	Amersfoortsestraat 20, Soesterberg

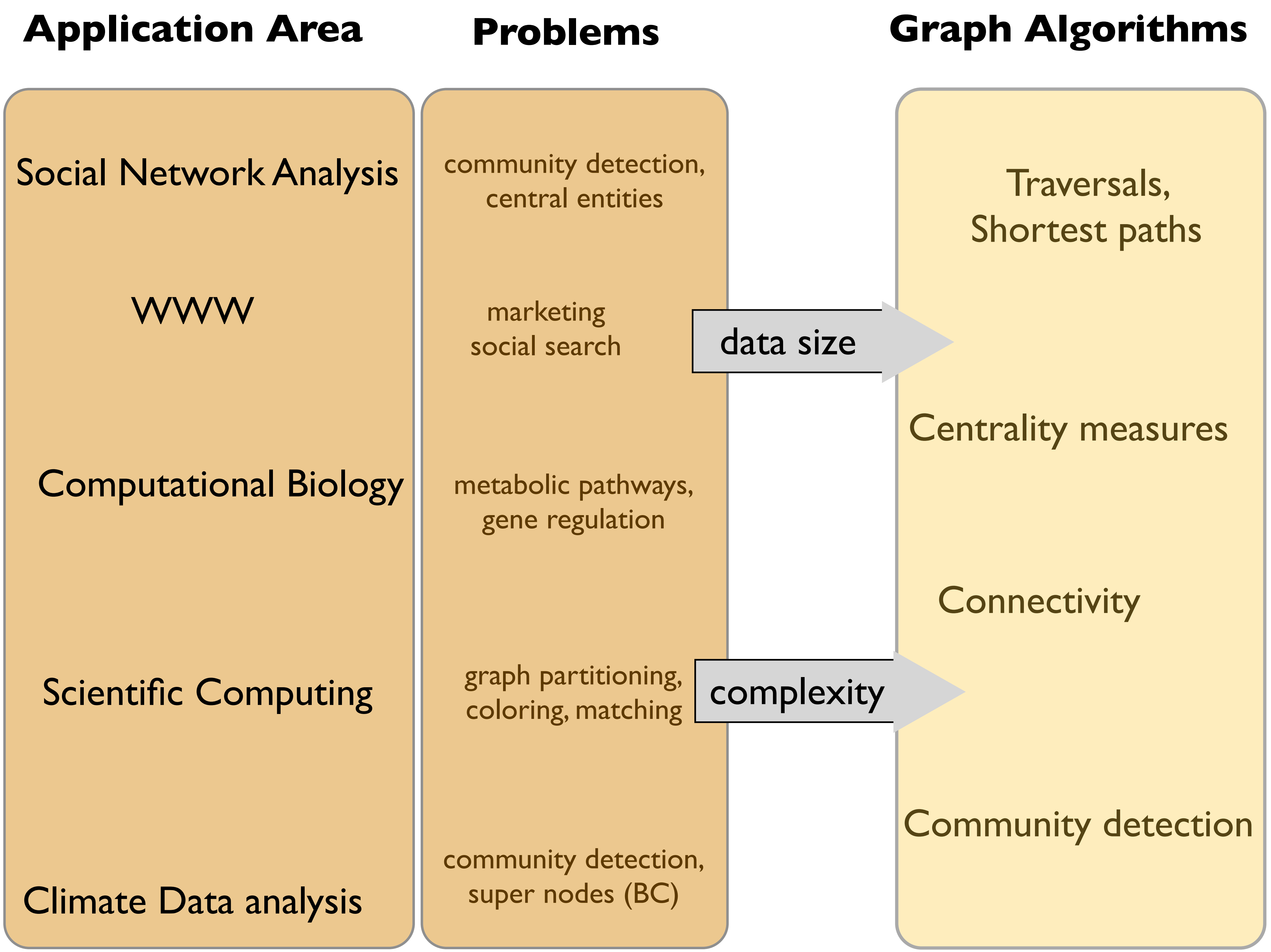
Miguel Angel Bermejo, Geliual Lauieg, Victor Rodriguez and 5 others like this. ✓ Seen by 11

Brt Ilg thanks a lot!!!
April 15 at 3:38pm · Like

Qingyi Cherry Feng My pleasure 🙂
April 15 at 3:38pm · Like

Veronika Stolbova Thank you!=)
April 15 at 9:49pm · Like

Graph Analysis
“from Domain-Specific requirements, to Computation…”



Application Area

Social Network Analysis

WWW

Computational Biology

Scientific Computing

Climate Data analysis

Problems

community detection,
central entities

marketing
social search

metabolic pathways,
gene regulation

graph partitioning,
coloring, matching

community detection,
super nodes (BC)

data size

complexity

Graph Algorithms

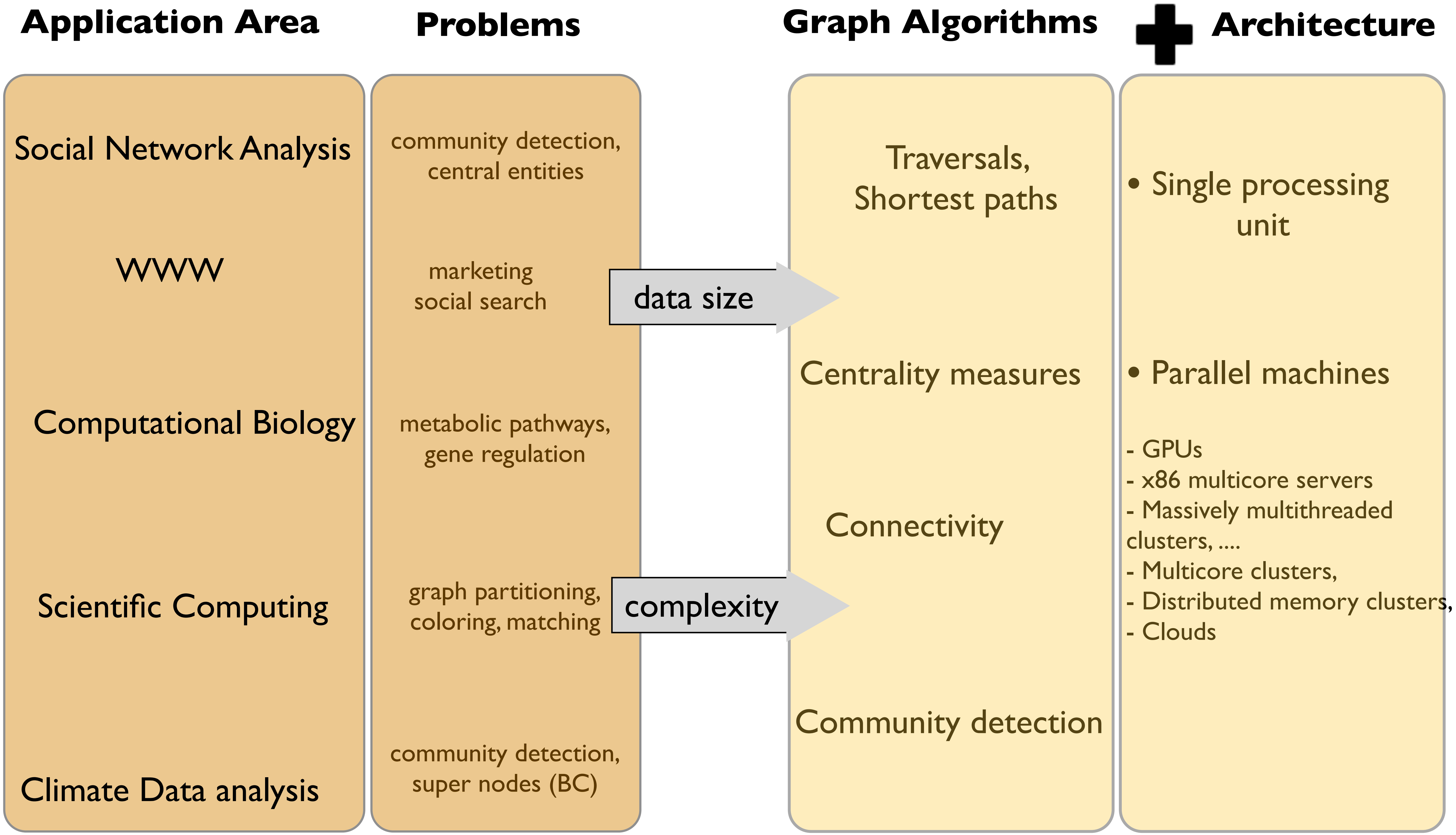
Traversals,
Shortest paths

Centrality measures

Connectivity

Community detection

Graph Analysis
“from Domain-Specific requirements, to Computation...”



Application Area

Social Network Analysis

WWW

Computational Biology

Scientific Computing

Climate Data analysis

Problems

community detection, central entities

marketing social search

metabolic pathways, gene regulation

graph partitioning, coloring, matching

community detection, super nodes (BC)

data size

complexity

Graph Algorithms

Traversals, Shortest paths

Centrality measures

Connectivity

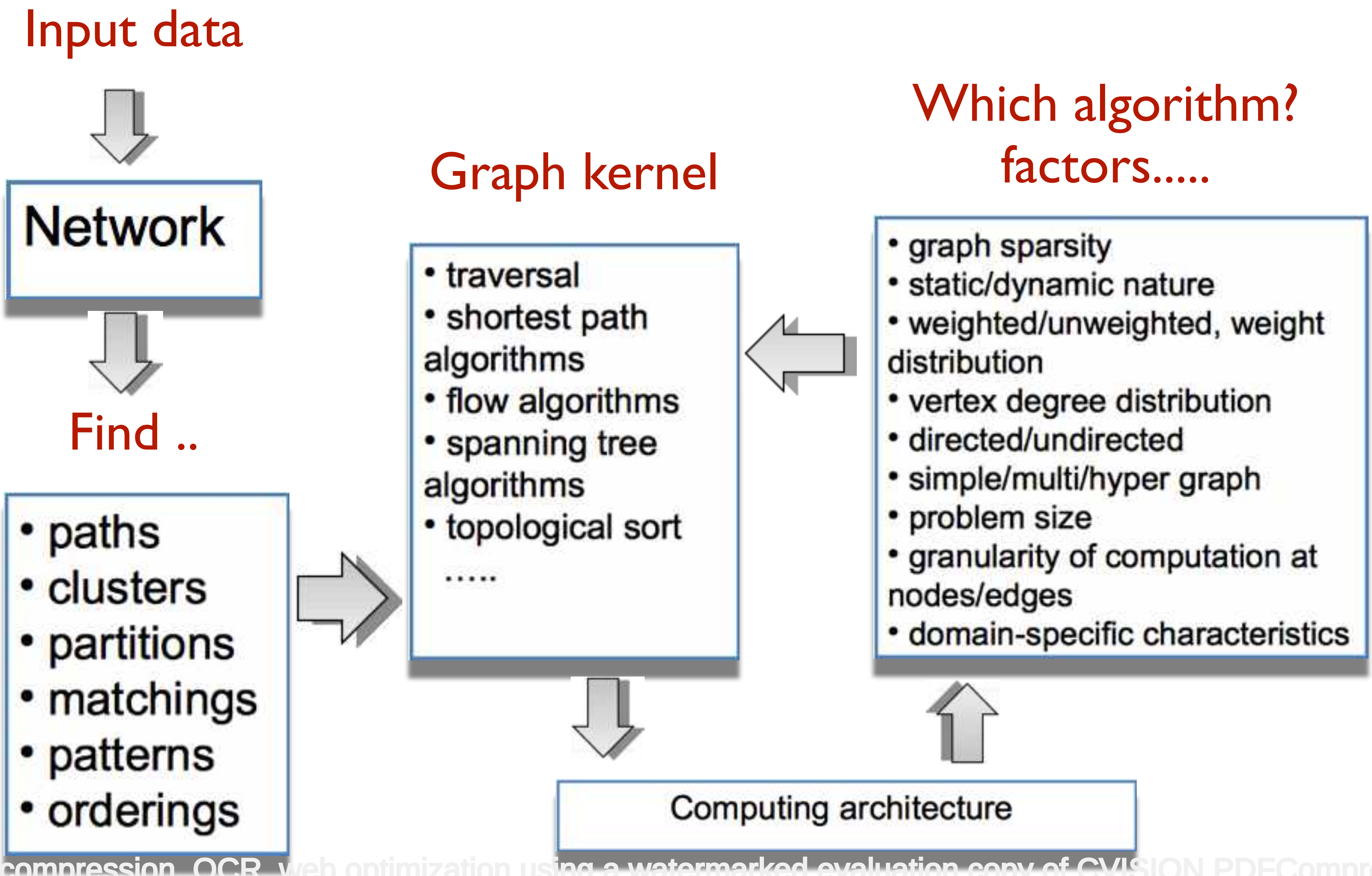
Community detection

Architecture

- Single processing unit
- Parallel machines

- GPUs
- x86 multicore servers
- Massively multithreaded clusters, ...
- Multicore clusters,
- Distributed memory clusters,
- Clouds

Graph Analysis
“from Domain-Specific requirements, to Computation…”



Input data

Network

Find ..

- paths
- clusters
- partitions
- matchings
- patterns
- orderings

Graph kernel

- traversal
- shortest path algorithms
- flow algorithms
- spanning tree algorithms
- topological sort

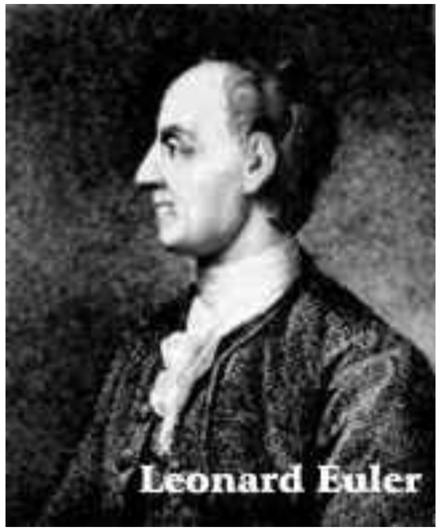
.....

Which algorithm? factors.....

- graph sparsity
- static/dynamic nature
- weighted/unweighted, weight distribution
- vertex degree distribution
- directed/undirected
- simple/multi/hyper graph
- problem size
- granularity of computation at nodes/edges
- domain-specific characteristics

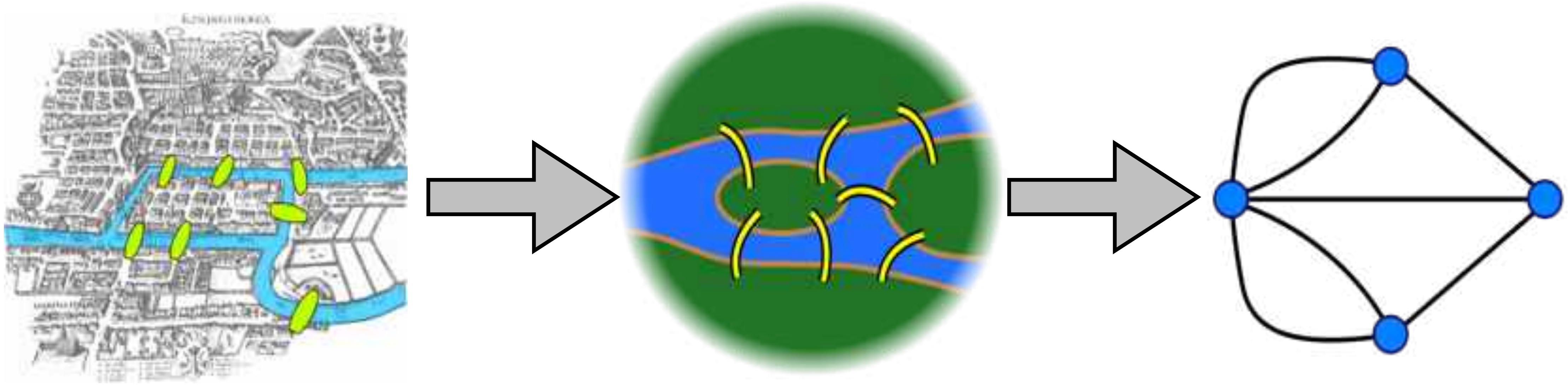
Computing architecture

Graph Theory: Review



- ➡ Began in 1735
- ➡ Bridges of Königsberg (today's Kaliningrad)

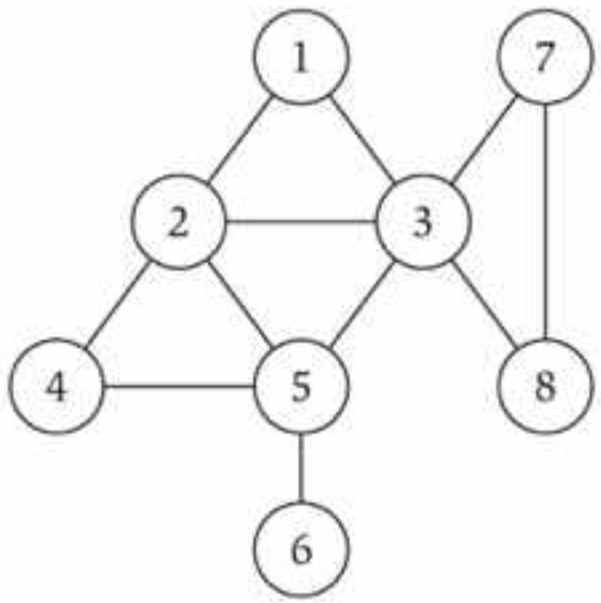
walk all 7 bridges without crossing each of them **ONCE**



Euler's solution of the Königsberg bridge problem is considered to be the first theorem of **graph theory**.

Graph Theory: Review

- **Graph** – mathematical object consisting of a set of:
  - ○ V = **nodes** (vertices, points).
  - ○ E = **edges** (links, arcs) between pairs of nodes.
  - ○ Denoted by $G = (V, E)$.
  - ○ Captures pairwise relationship between objects.
  - ○ **Graph size** parameters: $n = |V|$, $m = |E|$.



V = { 1, 2, 3, 4, 5, 6, 7, 8 }

E = { {1,2}, {1,3}, {2,3}, {2,4}, {2,5}, {3,5}, {3,7}, {3,8}, {4,5}, {5,6} }

n = 8

m = 11

Graph Theory: Networks and Analysis

- Types (Topological features)
  - degree distribution
  - clustering coefficient
  - assortativity, comm. structure,
  - hierarchical structure

a) “Simple” networks

- Lattices, random graphs, ...

b) Complex networks

- Scale-Free: power-law degree distribution (heavy-tailed DD)
- Small-World: small diameter and a high clustering coefficient.

- Analysis

1-Element-Level

- degree, betweenness, closeness centralities.

2-Group-Level

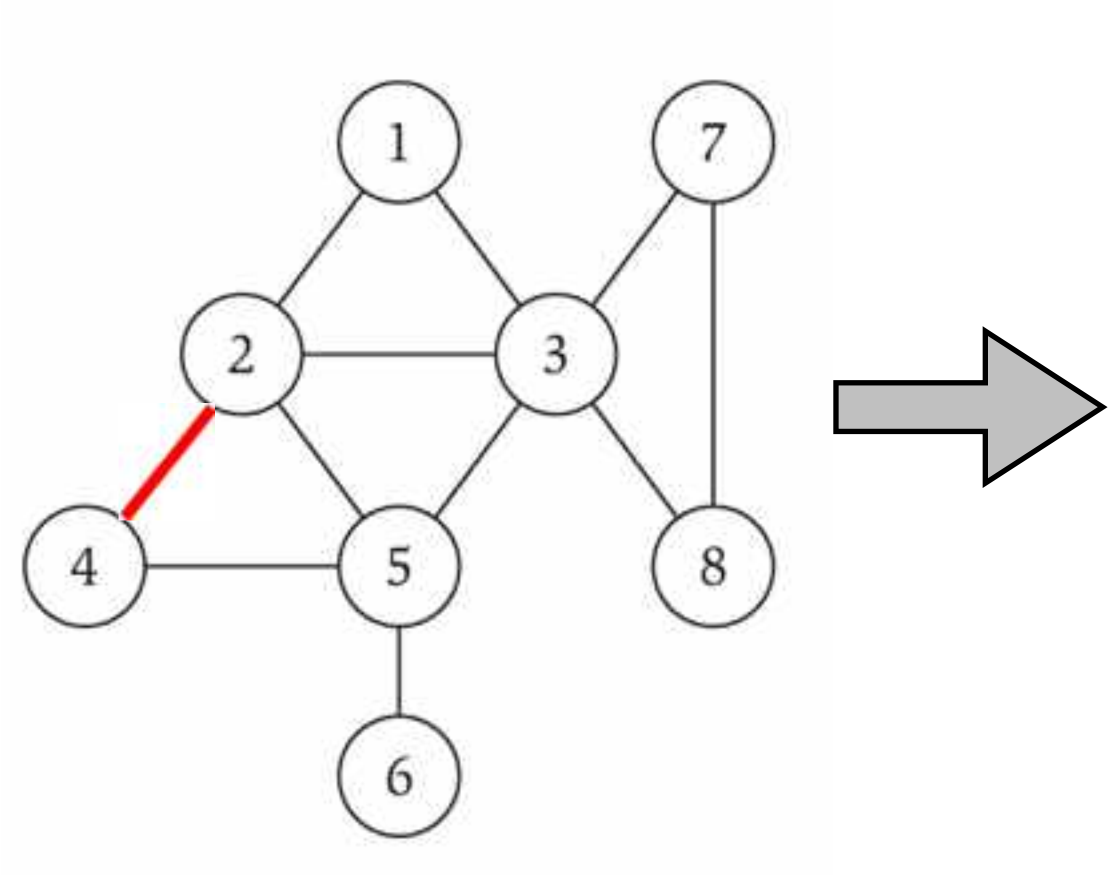
- network clustering (communities)

3-Network-Level

- small world effect (network diameter), transitivity

Graph Theory:
Graph (Network) Representation: “Data Structure”

- **Adjacency matrix.** n-by-n matrix with A_{uv} = 1 if (u, v) is an edge.
  - Two representations of each edge (symmetric matrix for undirected graphs; not for directed graphs).
  - space? Space: proportional to n^2.



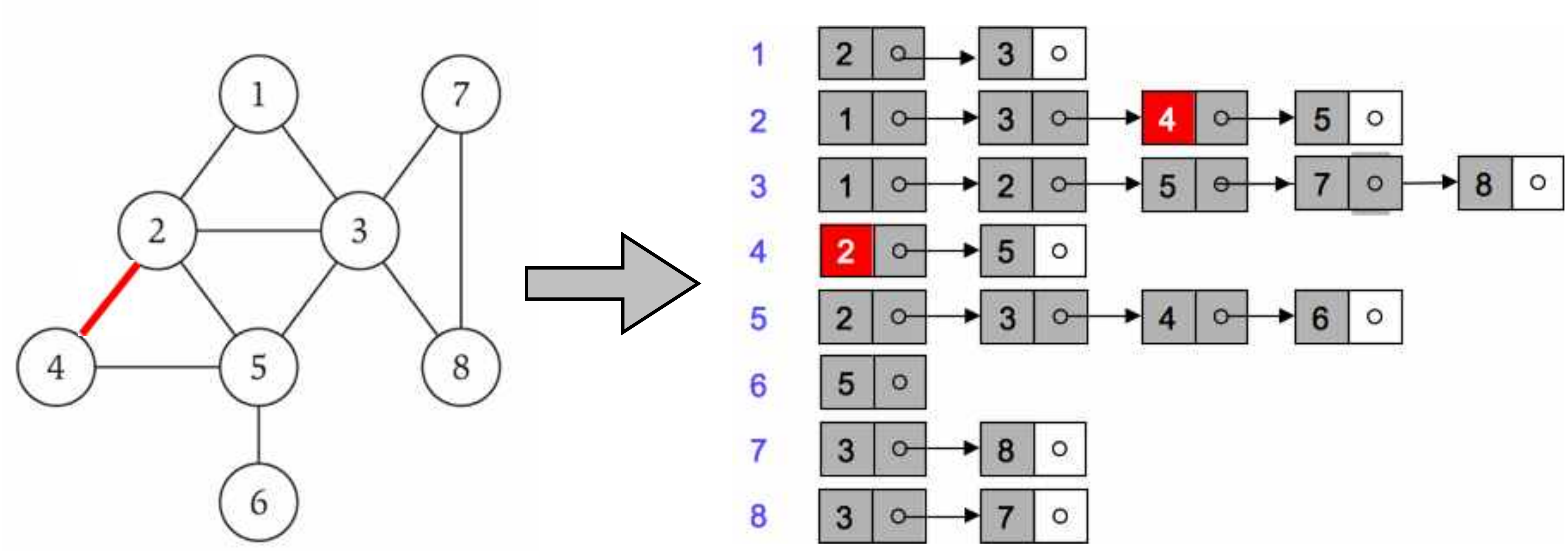
	1	2	3	4	5	6	7	8
1	0	1	1	0	0	0	0	0
2	1	0	1	1	1	0	0	0
3	1	1	0	0	1	0	1	1
4	0	1	0	1	1	0	0	0
5	0	1	1	1	0	1	0	0
6	0	0	0	0	1	0	0	0
7	0	0	1	0	0	0	0	1
8	0	0	1	0	0	0	1	0

Graph Theory: Review

Graph Representation: “Data Structure”

- **Adjacency list.** Node indexed array of lists.
  - Two representations of each edge.
  - Space proportional to $m + n$.

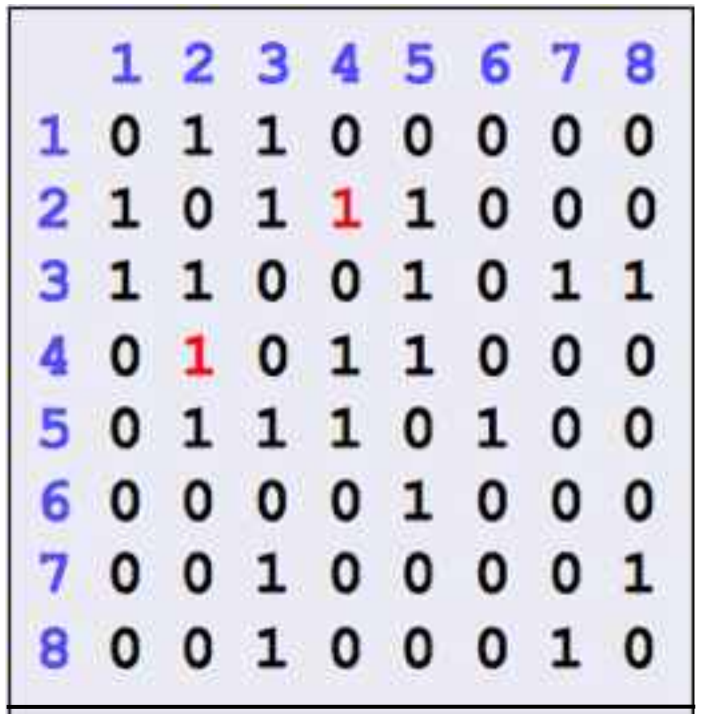
space?



“Data Structure”: Why it matters??

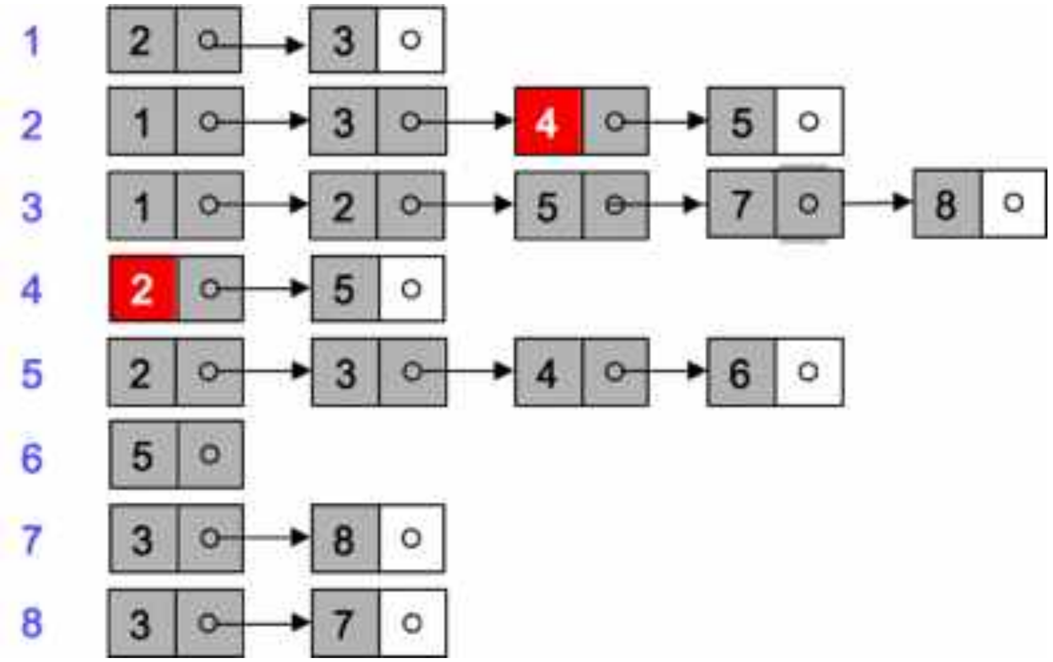
space and **complexity** tradeoffs?

Adjacency matrix



	1	2	3	4	5	6	7	8
1	0	1	1	0	0	0	0	0
2	1	0	1	1	1	0	0	0
3	1	1	0	0	1	0	1	1
4	0	1	0	1	1	0	0	0
5	0	1	1	1	0	1	0	0
6	0	0	0	0	1	0	0	0
7	0	0	1	0	0	0	0	1
8	0	0	1	0	0	0	1	0

Adjacency list



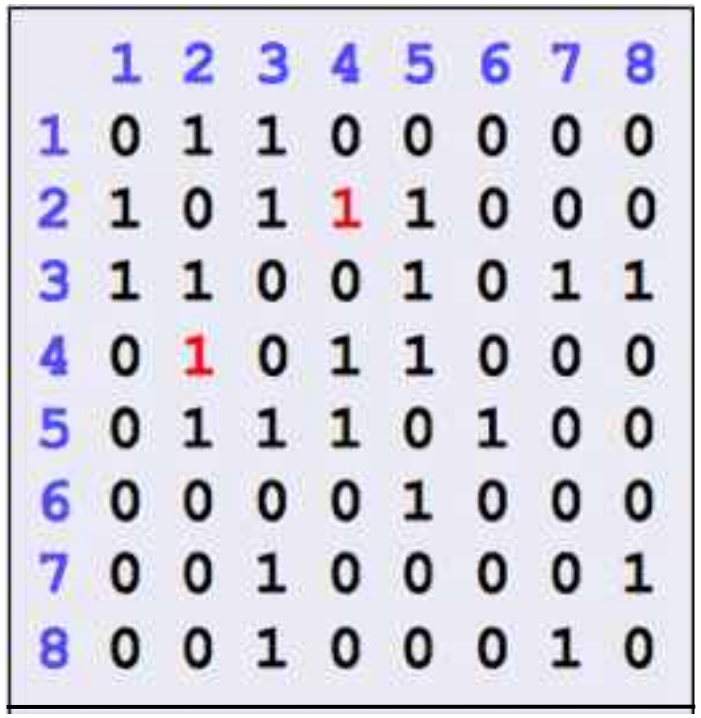
- ➡ An adjacency matrix uses O(n*n) memory. It has **fast lookups to check for presence or absence of a specific edge O(1)**, but slow to iterate over all edges.

- ➡ Adjacency lists use memory in proportion to the number edges, which might save a lot of memory if the graph is sparse. It is fast to **iterate over all edges**, but finding the presence or absence of specific edge is slower than with the matrix.

“Data Structure”: Why it matters??

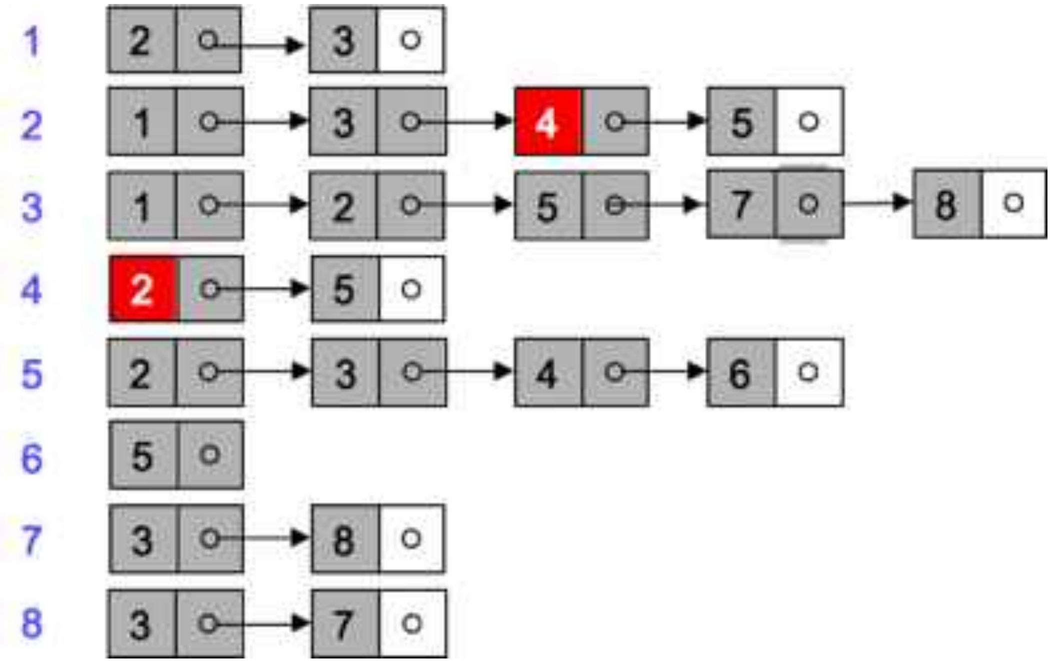
space and **complexity** tradeoffs?

Adjacency matrix



	1	2	3	4	5	6	7	8
1	0	1	1	0	0	0	0	0
2	1	0	1	1	1	0	0	0
3	1	1	0	0	1	0	1	1
4	0	1	0	1	1	0	0	0
5	0	1	1	1	0	1	0	0
6	0	0	0	0	1	0	0	0
7	0	0	1	0	0	0	0	1
8	0	0	1	0	0	0	1	0

Adjacency list



There is often a **time-space-tradeoff** involved in such problems, that is, it cannot be solved with few computing time *and* low memory consumption. One then has to make a compromise and to exchange computing time for memory consumption or vice versa, depending on which algorithm one chooses and how one parameterizes it.

Algorithm Complexity
Running Time for Algorithms: e.g.,

Give me the number of edges in a graph using the adjacency matrix as a data structure!!

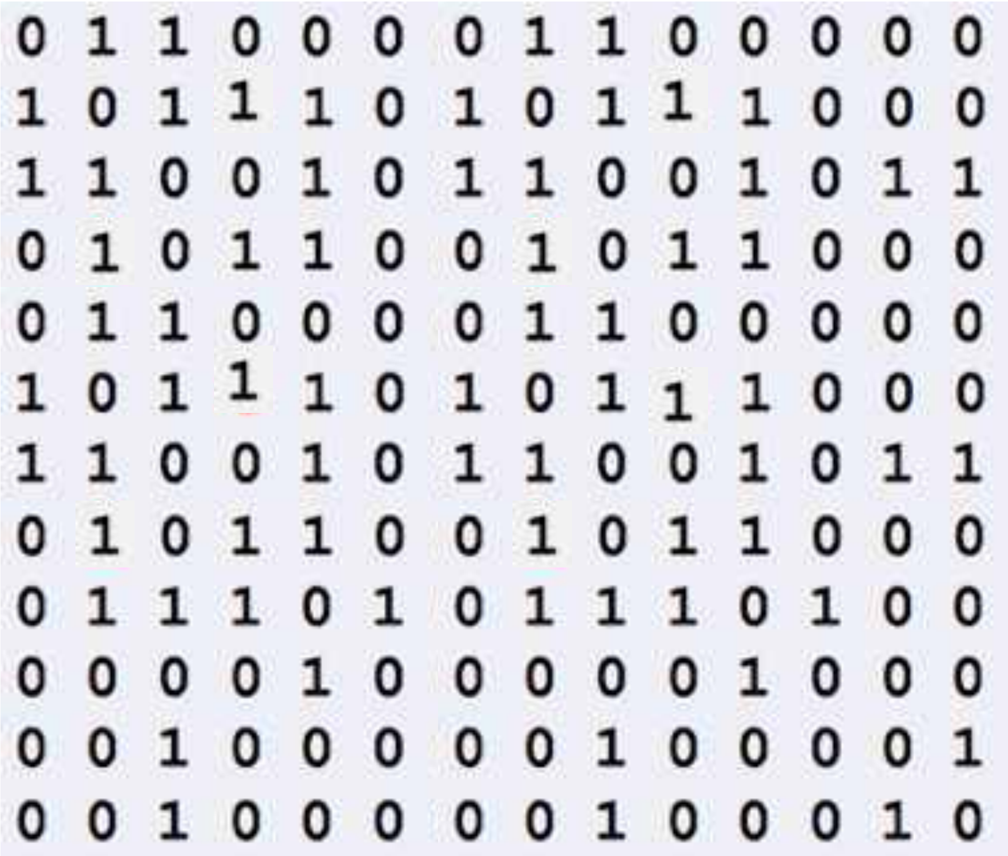
Algorithm

```
edges = 0;

for (i = 0; i < n; i++) {
  for (j = 0; j < n; j++) {
        if ($A_{ij}$ == 1)
          edges=edges+1;
    }
}
    return edges;
```

Runtime = O(n^2)

Adjacency matrix A



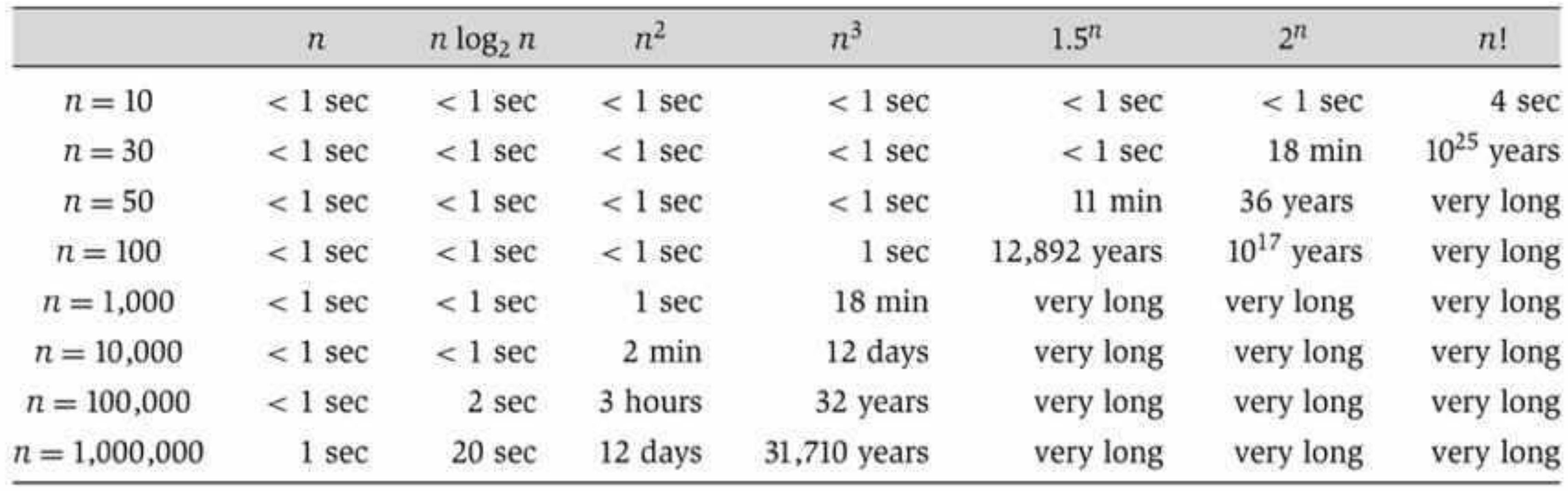
0	1	1	0	0	0	0	1	1	0	0	0	0	0
1	0	1	1	1	0	1	0	1	1	1	0	0	0
1	1	0	0	1	0	1	1	0	0	1	0	1	1
0	1	0	1	1	0	0	1	0	1	1	0	0	0
0	1	1	0	0	0	0	1	1	0	0	0	0	0
1	0	1	1	1	0	1	0	1	1	1	0	0	0
1	1	0	0	1	0	1	1	0	0	1	0	1	1
0	1	0	1	1	0	0	1	0	1	1	0	0	0
0	1	1	1	0	1	0	1	1	1	0	1	0	0
0	0	0	0	1	0	0	0	0	0	1	0	0	0
0	0	1	0	0	0	0	0	1	0	0	0	0	1
0	0	1	0	0	0	0	0	1	0	0	0	1	0

G no of nodes=n

Algorithm Complexity
Running Time for Algorithms: e.g.,

Complexity using **big-O notation** (run time). For a problem of size N:

- a constant-time method is "order 1": O(1)
- a linear-time method is "order N": O(N)
- a quadratic-time method is "order N squared": O(N^2)



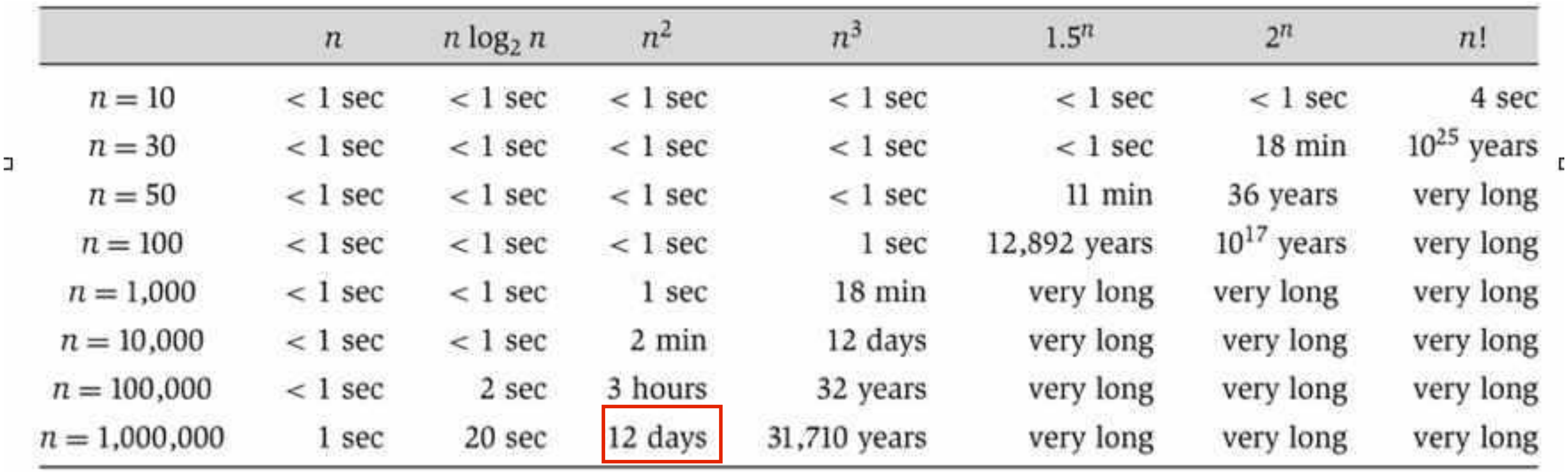
	n	$n \log_2 n$	n^2	n^3	1.5^n	2^n	$n!$
$n = 10$	< 1 sec	< 1 sec	< 1 sec	< 1 sec	< 1 sec	< 1 sec	4 sec
$n = 30$	< 1 sec	< 1 sec	< 1 sec	< 1 sec	< 1 sec	18 min	10^{25} years
$n = 50$	< 1 sec	< 1 sec	< 1 sec	< 1 sec	11 min	36 years	very long
$n = 100$	< 1 sec	< 1 sec	< 1 sec	1 sec	12,892 years	10^{17} years	very long
$n = 1{,}000$	< 1 sec	< 1 sec	1 sec	18 min	very long	very long	very long
$n = 10{,}000$	< 1 sec	< 1 sec	2 min	12 days	very long	very long	very long
$n = 100{,}000$	< 1 sec	2 sec	3 hours	32 years	very long	very long	very long
$n = 1{,}000{,}000$	1 sec	20 sec	12 days	31,710 years	very long	very long	very long

The running times (rounded up) of different algorithms on inputs of increasing size, for a processor performing a **million high-level instructions** per second, in cases where running time exceeds 10^{25} years, "very long time" is recorded.

Algorithm Complexity
Running Time for Algorithms: e.g.,

Complexity using **big-O notation** (run time). For a problem of size N:

- a constant-time method is "order 1": O(1)
- a linear-time method is "order N": O(N)
- a quadratic-time method is "order N squared": O(N^2)



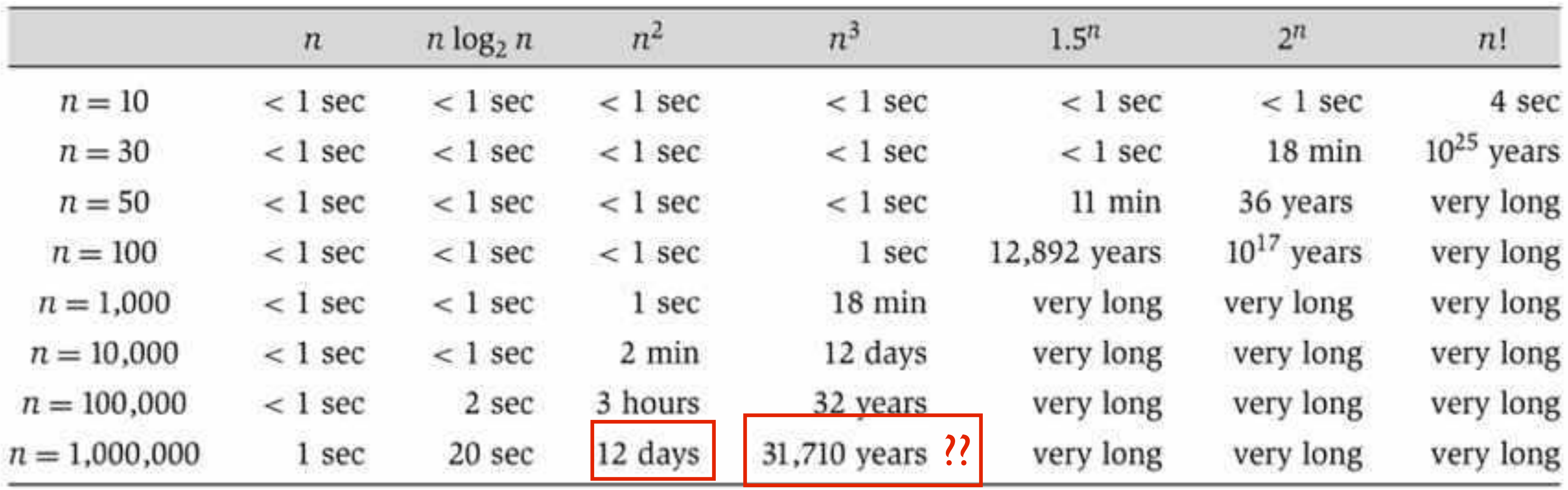
	n	$n \log_2 n$	n^2	n^3	1.5^n	2^n	$n!$
$n = 10$	< 1 sec	< 1 sec	< 1 sec	< 1 sec	< 1 sec	< 1 sec	4 sec
$n = 30$	< 1 sec	< 1 sec	< 1 sec	< 1 sec	< 1 sec	18 min	10^{25} years
$n = 50$	< 1 sec	< 1 sec	< 1 sec	< 1 sec	11 min	36 years	very long
$n = 100$	< 1 sec	< 1 sec	< 1 sec	1 sec	12,892 years	10^{17} years	very long
$n = 1{,}000$	< 1 sec	< 1 sec	1 sec	18 min	very long	very long	very long
$n = 10{,}000$	< 1 sec	< 1 sec	2 min	12 days	very long	very long	very long
$n = 100{,}000$	< 1 sec	2 sec	3 hours	32 years	very long	very long	very long
$n = 1{,}000{,}000$	1 sec	20 sec	12 days	31,710 years	very long	very long	very long

The running times (rounded up) of different algorithms on inputs of increasing size, for a processor performing a **million high-level instructions** per second, in cases where running time exceeds 10^{25} years, "very long time" is recorded.

Algorithm Complexity
Running Time for Algorithms: e.g.,

Complexity using **big-O notation** (run time). For a problem of size N:

- a constant-time method is "order 1": O(1)
- a linear-time method is "order N": O(N)
- a quadratic-time method is "order N squared": O(N^2)

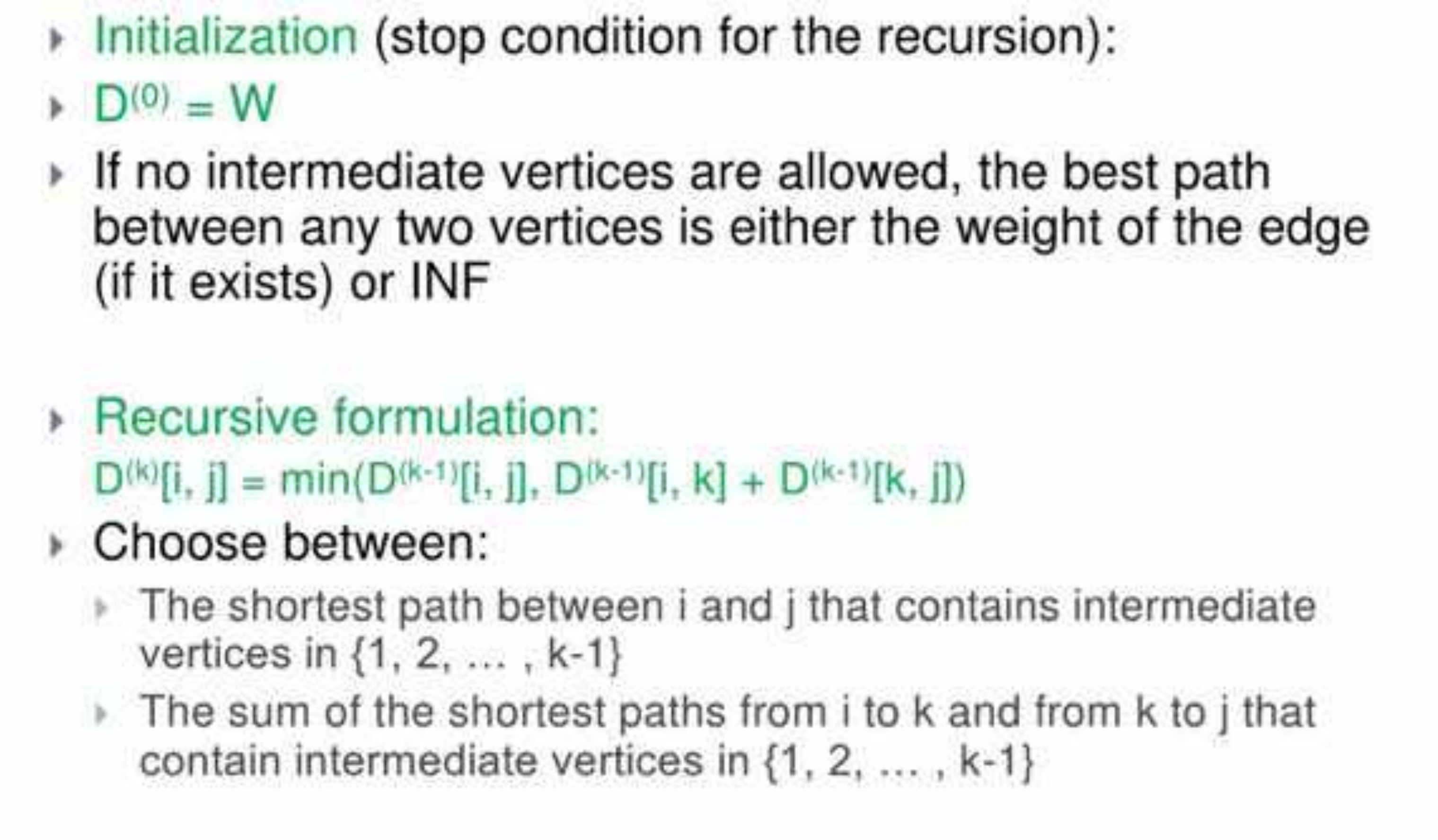


	n	$n \log_2 n$	n^2	n^3	1.5^n	2^n	$n!$
$n = 10$	< 1 sec	< 1 sec	< 1 sec	< 1 sec	< 1 sec	< 1 sec	4 sec
$n = 30$	< 1 sec	< 1 sec	< 1 sec	< 1 sec	< 1 sec	18 min	10^{25} years
$n = 50$	< 1 sec	< 1 sec	< 1 sec	< 1 sec	11 min	36 years	very long
$n = 100$	< 1 sec	< 1 sec	< 1 sec	1 sec	12,892 years	10^{17} years	very long
$n = 1{,}000$	< 1 sec	< 1 sec	1 sec	18 min	very long	very long	very long
$n = 10{,}000$	< 1 sec	< 1 sec	2 min	12 days	very long	very long	very long
$n = 100{,}000$	< 1 sec	2 sec	3 hours	32 years	very long	very long	very long
$n = 1{,}000{,}000$	1 sec	20 sec	12 days	31,710 years ??	very long	very long	very long

The running times (rounded up) of different algorithms on inputs of increasing size, for a processor performing a **million high-level instructions** per second, in cases where running time exceeds 10^{25} years, "very long time" is recorded.

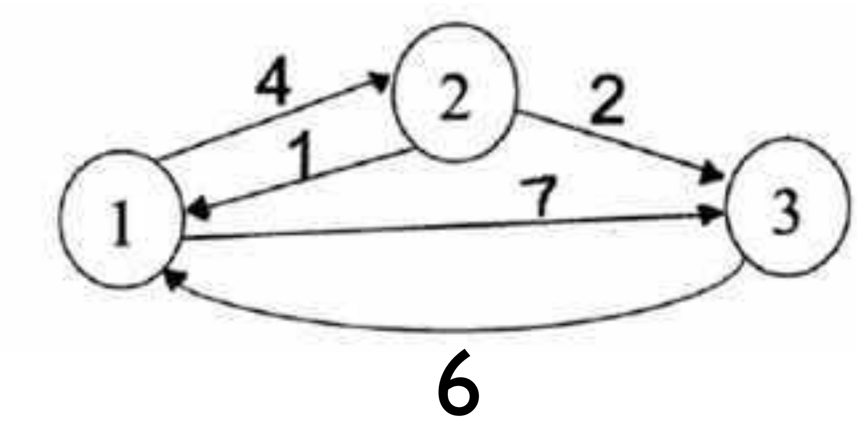
Algorithm Complexity

Network “Shortest Paths”: Floyd–Warshall algorithm

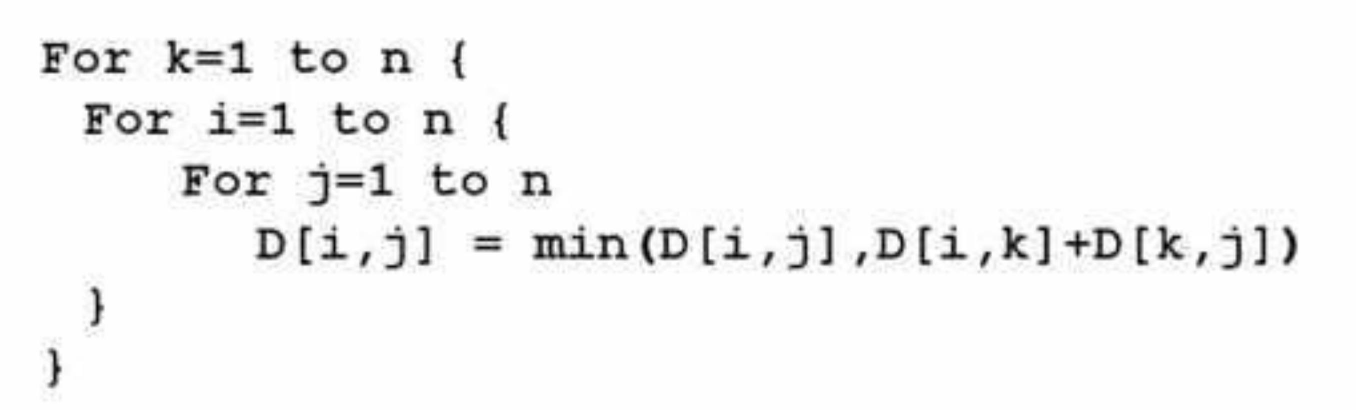


- Initialization (stop condition for the recursion):
- $D^{(0)} = W$
- If no intermediate vertices are allowed, the best path between any two vertices is either the weight of the edge (if it exists) or INF

- Recursive formulation:
  $D^{(k)}[i, j] = \min(D^{(k-1)}[i, j], D^{(k-1)}[i, k] + D^{(k-1)}[k, j])$
- Choose between:
  - The shortest path between i and j that contains intermediate vertices in {1, 2, … , k-1}
  - The sum of the shortest paths from i to k and from k to j that contain intermediate vertices in {1, 2, … , k-1}

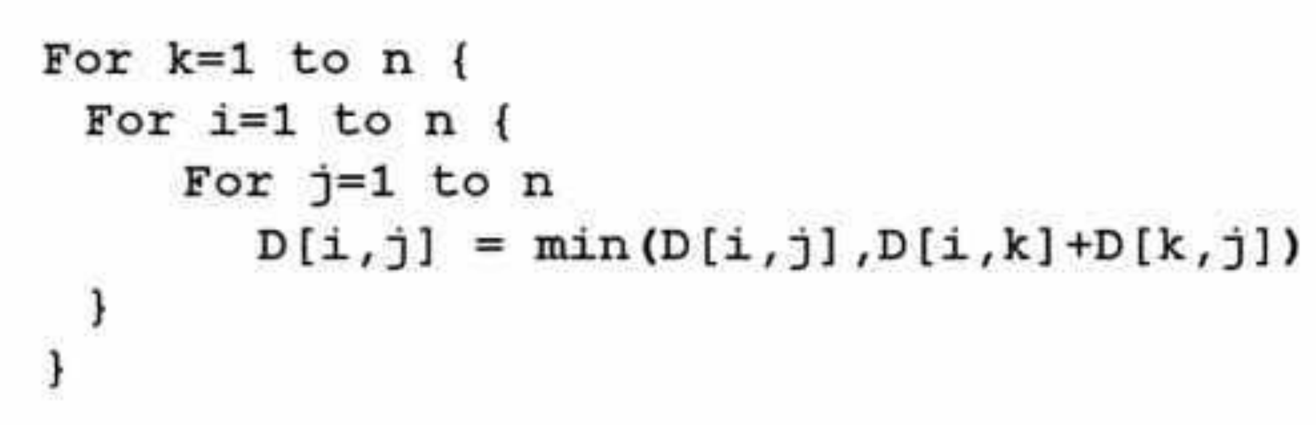


$$D^{(0)} \begin{bmatrix} 0 & 4 & 7 \\ 1 & 0 & 2 \\ 6 & \infty & 0 \end{bmatrix}$$

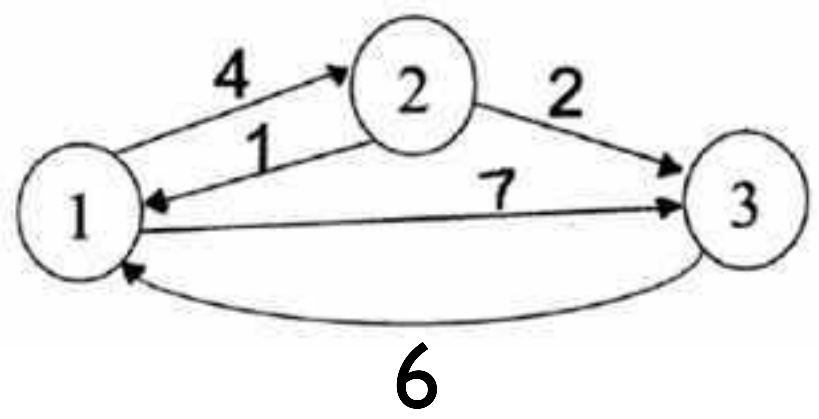


```
For k=1 to n {
  For i=1 to n {
      For j=1 to n
          D[i,j] = min(D[i,j],D[i,k]+D[k,j])
  }
}
```

Graph Analyzing Algorithms:
Network “Shortest Paths”: Floyd–Warshall algorithm



```
For k=1 to n {
  For i=1 to n {
     For j=1 to n
        D[i,j] = min(D[i,j],D[i,k]+D[k,j])
  }
}
```



$$D^{(0)}\begin{bmatrix} 0 & 4 & 7 \\ 1 & 0 & 2 \\ 6 & \infty & 0 \end{bmatrix}$$

$$D^{(1)}\begin{bmatrix} 0 & 4 & 7 \\ 1 & 0 & 2 \\ 6 & \boxed{10} & 0 \end{bmatrix}$$

Consider Vertex 1:
D(3,2) = D(3,1) + D(1,2)

$$D^{(2)}\begin{bmatrix} 0 & 4 & \boxed{6} \\ 1 & 0 & 2 \\ 6 & 10 & 0 \end{bmatrix}$$

Consider Vertex 2:
D(1,3) = D(1,2) + D(2,3)

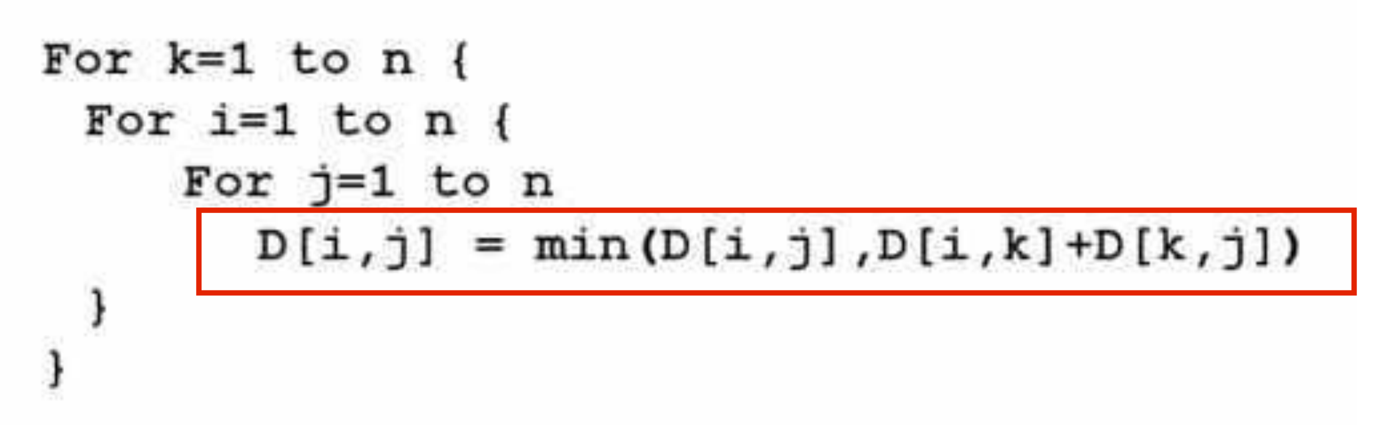
time: $O(n^3)$

Space: $O(n^2)$

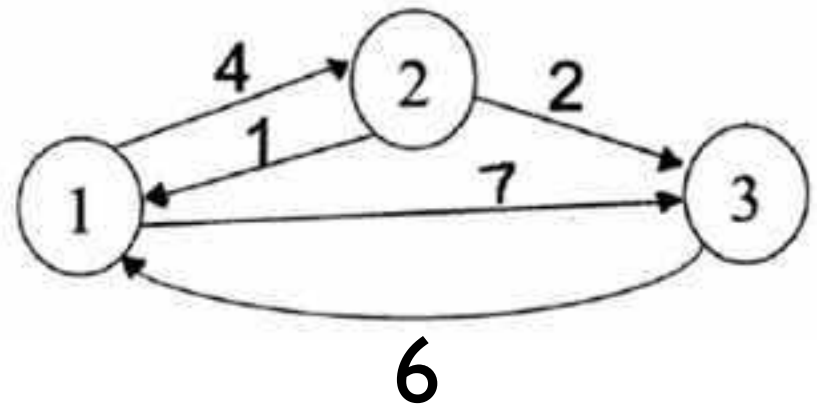
$$D^{(3)}\begin{bmatrix} 0 & 4 & 6 \\ 1 & 0 & 2 \\ 6 & 10 & 0 \end{bmatrix}$$

Consider Vertex 3:
Nothing changes.

Graph Analyzing Algorithms:
Network “Shortest Paths”: <u>Floyd–Warshall algorithm</u>



```
For k=1 to n {
  For i=1 to n {
     For j=1 to n
       D[i,j] = min(D[i,j],D[i,k]+D[k,j])
  }
}
```



$$D^{(0)}\begin{bmatrix} 0 & 4 & 7 \\ 1 & 0 & 2 \\ 6 & \infty & 0 \end{bmatrix}$$

$$D^{(1)}\begin{bmatrix} 0 & 4 & 7 \\ 1 & 0 & 2 \\ 6 & \boxed{10} & 0 \end{bmatrix}$$

<u>Consider Vertex 1:</u>
D(3,2) = D(3,1) + D(1,2)

$$D^{(2)}\begin{bmatrix} 0 & 4 & \boxed{6} \\ 1 & 0 & 2 \\ 6 & 10 & 0 \end{bmatrix}$$

<u>Consider Vertex 2:</u>
D(1,3) = D(1,2) + D(2,3)

$$D^{(3)}\begin{bmatrix} 0 & 4 & 6 \\ 1 & 0 & 2 \\ 6 & 10 & 0 \end{bmatrix}$$

<u>Consider Vertex 3:</u>
Nothing changes.

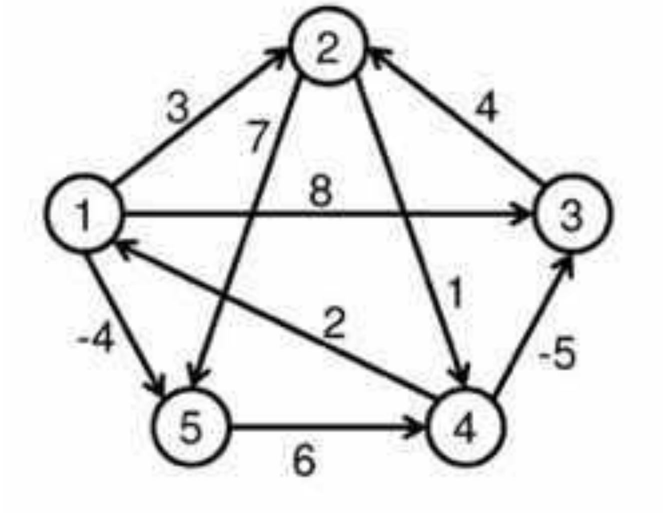
time: $O(n^3)$

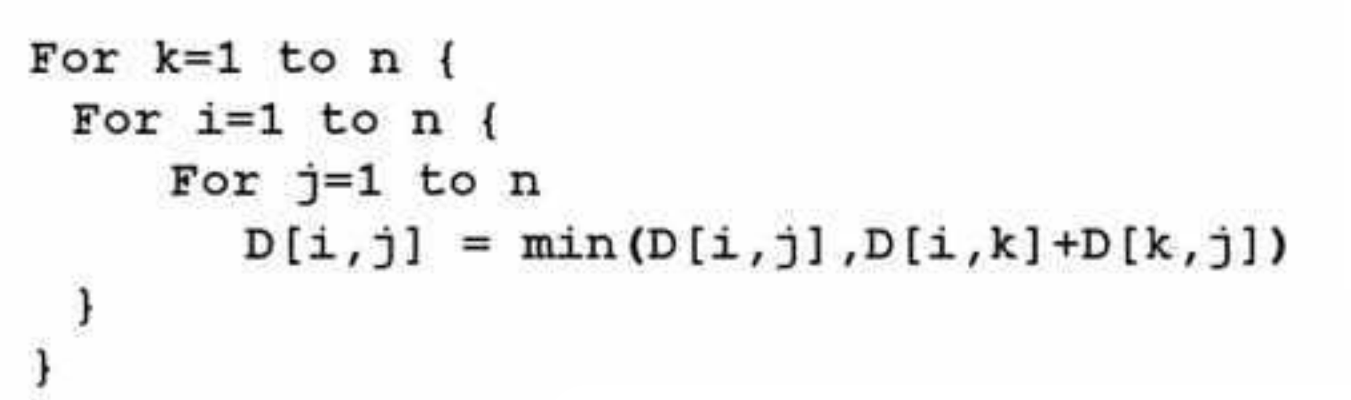
Space: $O(n^2)$

Algorithm Complexity
Betweenness Centrality

(same as shortest paths??)

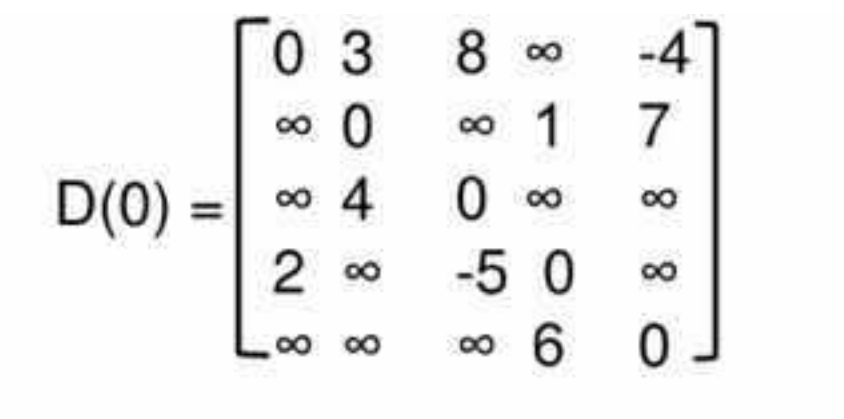
- Time: $O(N^3)$
- Space: $O(N^2)$





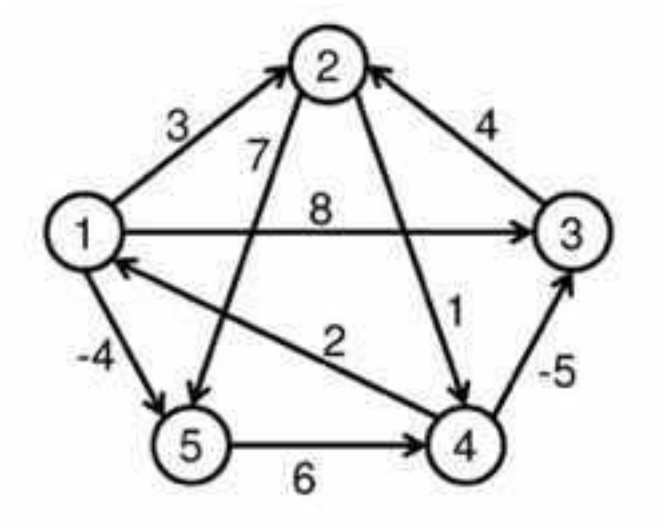
```
For k=1 to n {
 For i=1 to n {
    For j=1 to n
       D[i,j] = min(D[i,j],D[i,k]+D[k,j])
 }
}
```

D(0)



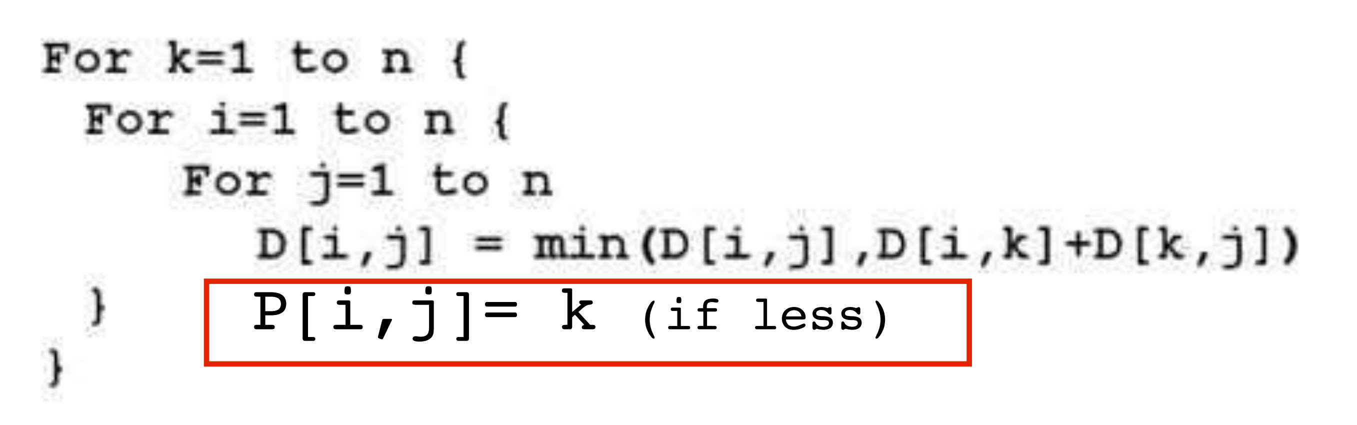
$$D(0) = \begin{bmatrix} 0 & 3 & 8 & \infty & -4 \\ \infty & 0 & \infty & 1 & 7 \\ \infty & 4 & 0 & \infty & \infty \\ 2 & \infty & -5 & 0 & \infty \\ \infty & \infty & \infty & 6 & 0 \end{bmatrix}$$

Algorithm Complexity
Betweenness Centrality



(same as shortest paths??)

- Time: $O(N^3)$
- Space: $O(N^2)$



```
For k=1 to n {
  For i=1 to n {
     For j=1 to n
        D[i,j] = min(D[i,j],D[i,k]+D[k,j])
  }     P[i,j]= k (if less)
}
```

D(0)

$$D(0) = \begin{bmatrix} 0 & 3 & 8 & \infty & -4 \\ \infty & 0 & \infty & 1 & 7 \\ \infty & 4 & 0 & \infty & \infty \\ 2 & \infty & -5 & 0 & \infty \\ \infty & \infty & \infty & 6 & 0 \end{bmatrix}$$

$$P(0) = \begin{bmatrix} \text{nil} & 1 & 1 & \text{nil} & 1 \\ \text{nil} & \text{nil} & \text{nil} & 2 & 2 \\ \text{nil} & 3 & \text{nil} & \text{nil} & \text{nil} \\ 4 & \text{nil} & 4 & \text{nil} & \text{nil} \\ \text{nil} & \text{nil} & \text{nil} & 5 & \text{nil} \end{bmatrix}$$

Algorithm Complexity
Graph Analyzing Algorithms

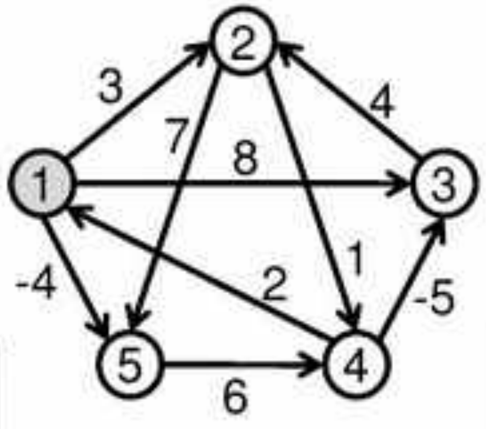
Betweenness Centrality
- Time: $O(N^3)$
- Space: $O(N^2)$

D(1)

$$D = \begin{bmatrix} 0 & 3 & 8 & \infty & -4 \\ \infty & 0 & \infty & 1 & 7 \\ \infty & 4 & 0 & \infty & \infty \\ 2 & \infty & -5 & 0 & \infty \\ \infty & \infty & \infty & 6 & 0 \end{bmatrix}$$

$$p = \begin{bmatrix} nil & 1 & 1 & nil & 1 \\ nil & nil & nil & 2 & 2 \\ nil & 3 & nil & nil & nil \\ 4 & nil & 4 & nil & nil \\ nil & nil & nil & 5 & nil \end{bmatrix}$$

D(0), P(0)



$$D = \begin{bmatrix} 0 & 3 & 8 & \infty & -4 \\ \infty & 0 & \infty & 1 & 7 \\ \infty & 4 & 0 & \infty & \infty \\ 2 & \boxed{5} & -5 & 0 & -2 \\ \infty & \infty & \infty & 6 & 0 \end{bmatrix}$$

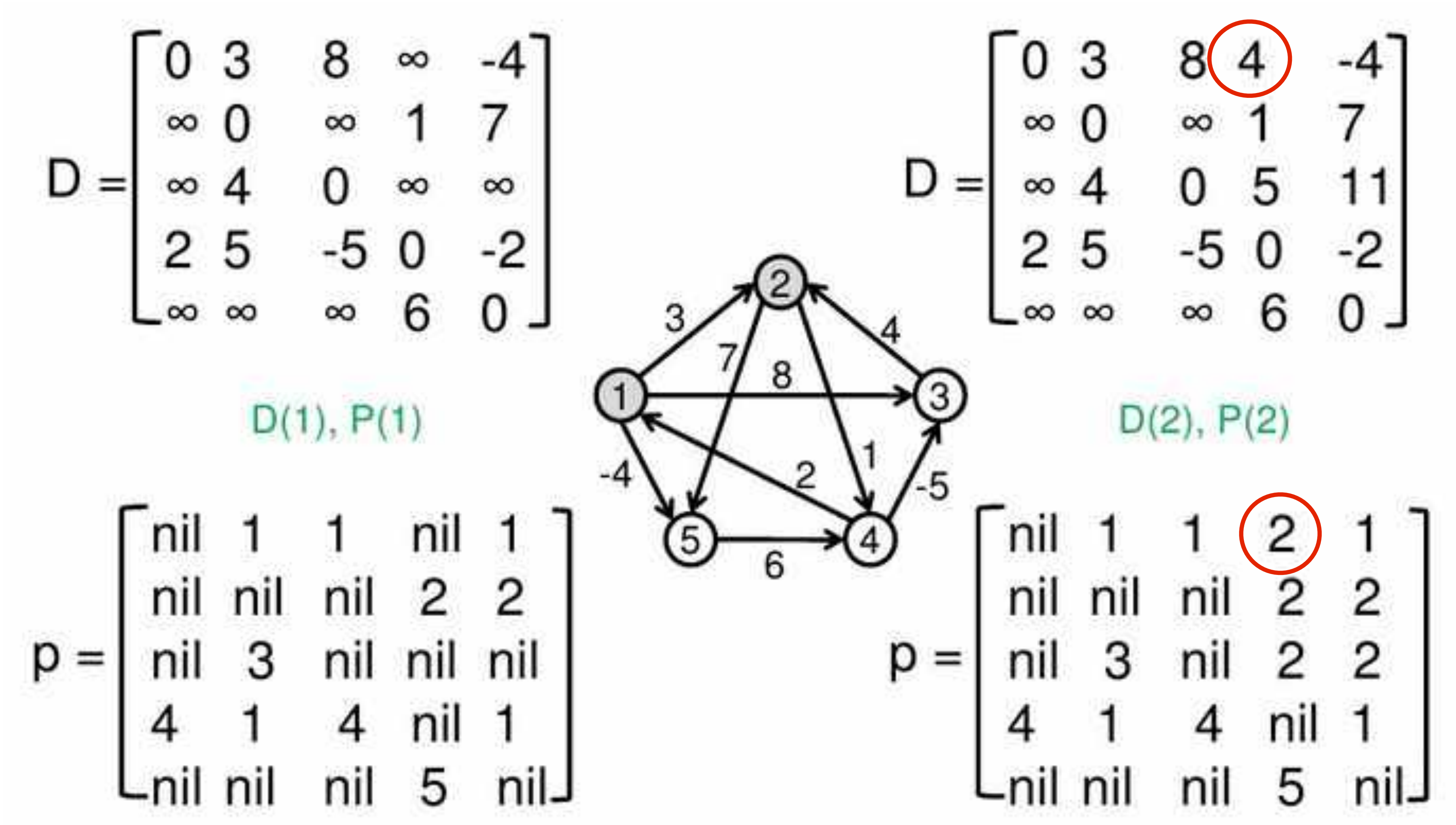
D(1), P(1)

$$p = \begin{bmatrix} nil & 1 & 1 & nil & 1 \\ nil & nil & nil & 2 & 2 \\ nil & 3 & nil & nil & nil \\ 4 & \boxed{1} & 4 & nil & 1 \\ nil & nil & nil & 5 & nil \end{bmatrix}$$

Algorithm Complexity
Graph Analyzing Algorithms

Betweenness Centrality
- Time: $O(N^3)$
- Space: $O(N^2)$

D(2)



$$D = \begin{bmatrix} 0 & 3 & 8 & \infty & -4 \\ \infty & 0 & \infty & 1 & 7 \\ \infty & 4 & 0 & \infty & \infty \\ 2 & 5 & -5 & 0 & -2 \\ \infty & \infty & \infty & 6 & 0 \end{bmatrix}$$

D(1), P(1)

$$p = \begin{bmatrix} \text{nil} & 1 & 1 & \text{nil} & 1 \\ \text{nil} & \text{nil} & \text{nil} & 2 & 2 \\ \text{nil} & 3 & \text{nil} & \text{nil} & \text{nil} \\ 4 & 1 & 4 & \text{nil} & 1 \\ \text{nil} & \text{nil} & \text{nil} & 5 & \text{nil} \end{bmatrix}$$

$$D = \begin{bmatrix} 0 & 3 & 8 & \boxed{4} & -4 \\ \infty & 0 & \infty & 1 & 7 \\ \infty & 4 & 0 & 5 & 11 \\ 2 & 5 & -5 & 0 & -2 \\ \infty & \infty & \infty & 6 & 0 \end{bmatrix}$$

D(2), P(2)

$$p = \begin{bmatrix} \text{nil} & 1 & 1 & \boxed{2} & 1 \\ \text{nil} & \text{nil} & \text{nil} & 2 & 2 \\ \text{nil} & 3 & \text{nil} & 2 & 2 \\ 4 & 1 & 4 & \text{nil} & 1 \\ \text{nil} & \text{nil} & \text{nil} & 5 & \text{nil} \end{bmatrix}$$

Algorithm Complexity
Graph Analyzing Algorithms

Betweenness Centrality

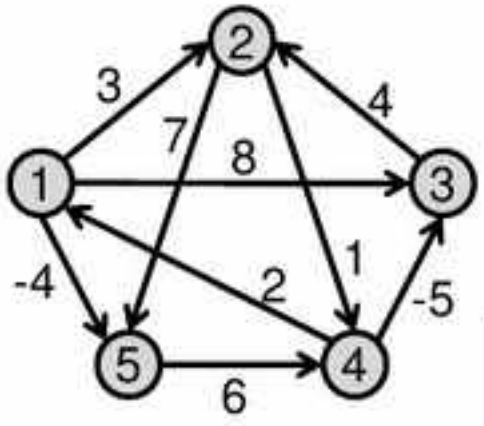
- Time: $O(N^3)$
- Space: $O(N^2)$

D(5)

$$D = \begin{bmatrix} 0 & 3 & -1 & 4 & -4 \\ 3 & 0 & -4 & 1 & -1 \\ 7 & 4 & 0 & 5 & 3 \\ 2 & -1 & -5 & 0 & -2 \\ 8 & 5 & 1 & 6 & 0 \end{bmatrix}$$

D(4), P(4)

$$p = \begin{bmatrix} \text{nil} & 1 & 4 & 2 & 1 \\ 4 & \text{nil} & 4 & 2 & 1 \\ 4 & 3 & \text{nil} & 2 & 1 \\ 4 & 3 & 4 & \text{nil} & 1 \\ 4 & 3 & 4 & 5 & \text{nil} \end{bmatrix}$$



$$D = \begin{bmatrix} 0 & 1 & -3 & 2 & -4 \\ 3 & 0 & -4 & 1 & -1 \\ 7 & 4 & 0 & 5 & 3 \\ 2 & -1 & -5 & 0 & -2 \\ 8 & 5 & 1 & 6 & 0 \end{bmatrix}$$

D(5), P(5)

$$p = \begin{bmatrix} \text{nil} & 3 & 4 & 5 & 1 \\ 4 & \text{nil} & 4 & 2 & 1 \\ 4 & 3 & \text{nil} & 2 & 1 \\ 4 & 3 & 4 & \text{nil} & 1 \\ 4 & 3 & 4 & 5 & \text{nil} \end{bmatrix}$$

Parallelism and HPC: “type with both hands, and ten fingers”

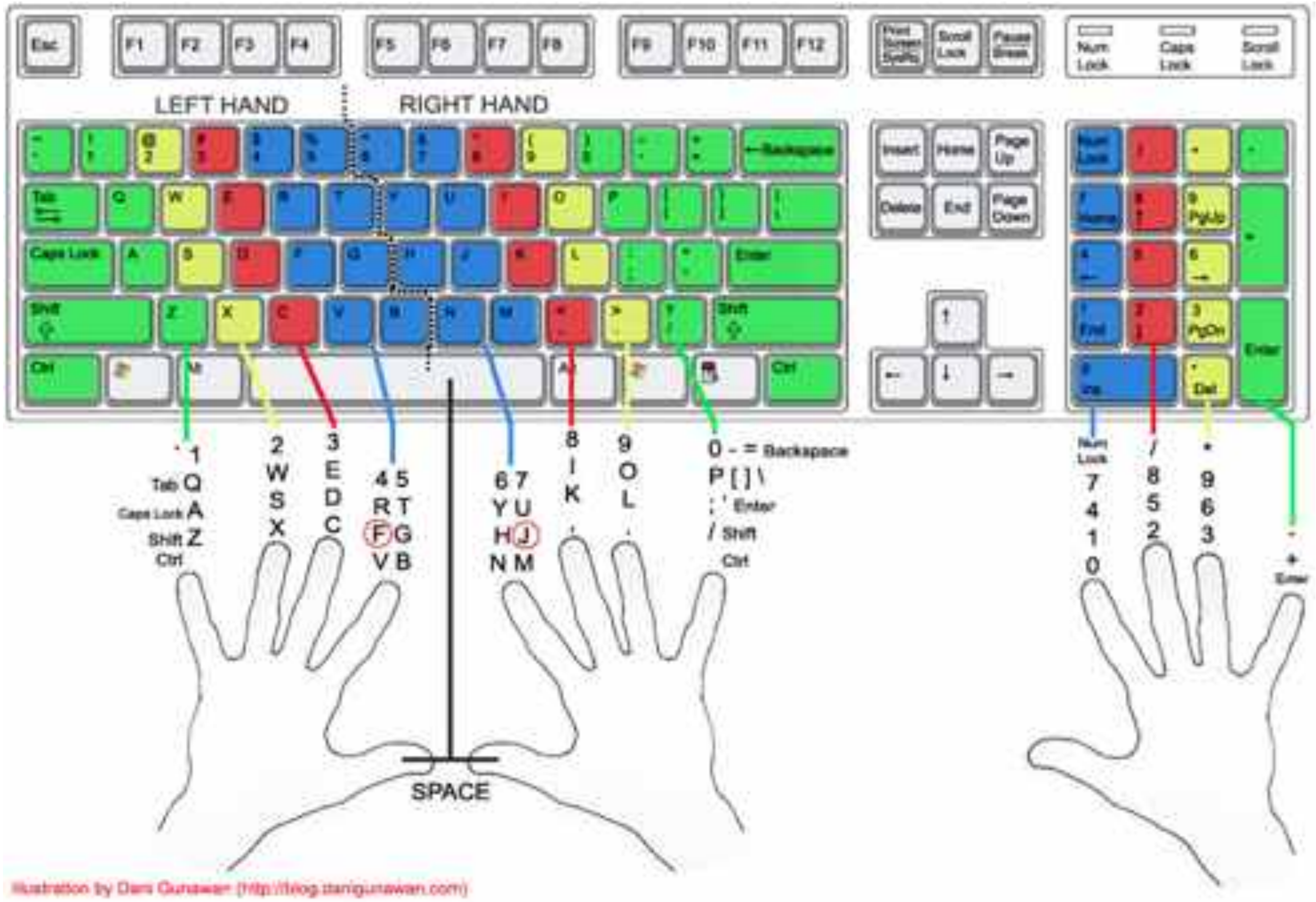
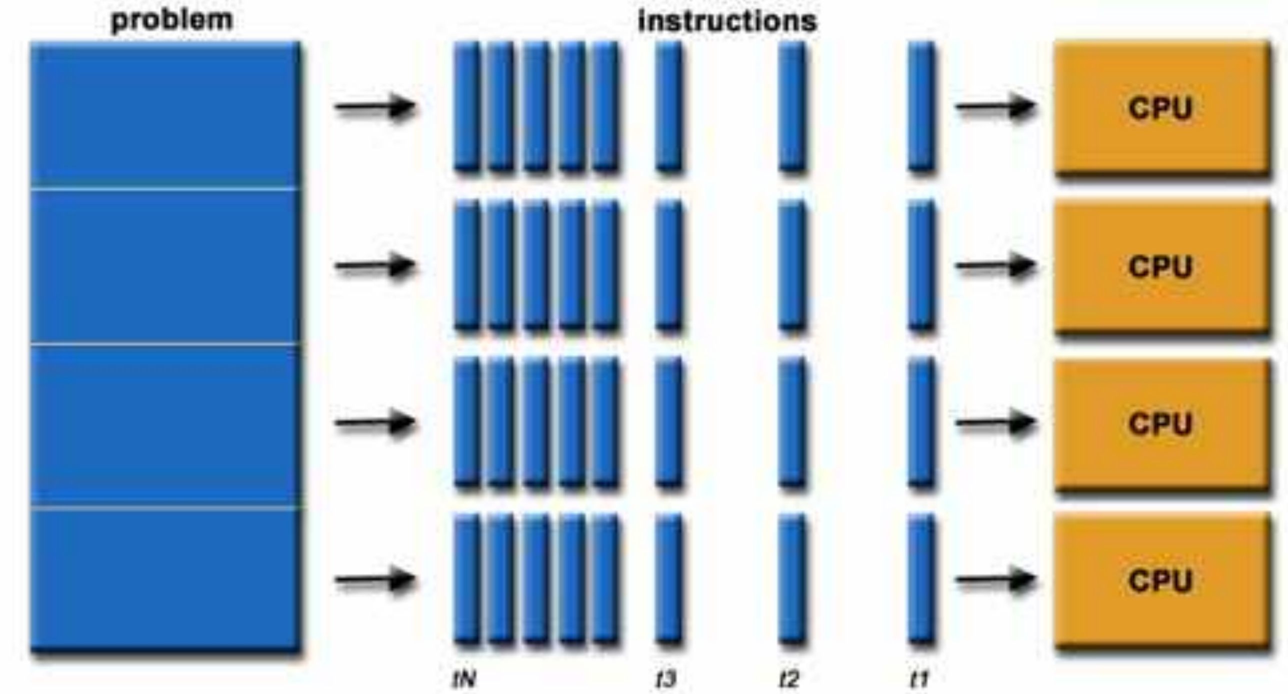
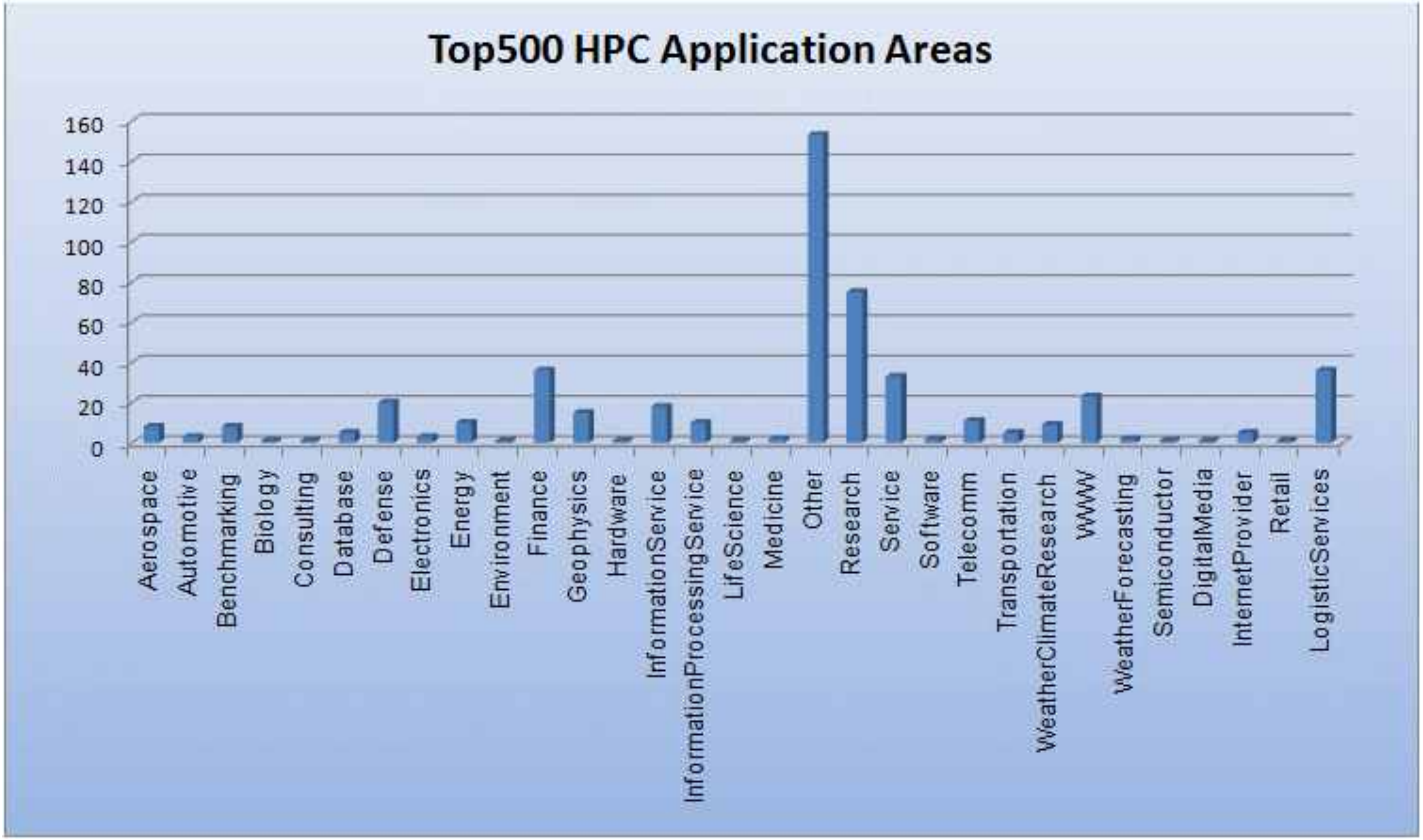


Illustration by Dani Gunawan (http://blog.danigunawan.com)



HPC: “Where?”


Top500 HPC Application Areas
160
140
120
100
80
60
40
20
0
Aerospace
Automotive
Benchmarking
Biology
Consulting
Database
Defense
Electronics
Energy
Environment
Finance
Geophysics
Hardware
InformationService
InformationProcessingService
LifeScience
Medicine
Other
Research
Service
Software
Telecomm
Transportation
WeatherClimateResearch
WWW
WeatherForecasting
Semiconductor
DigitalMedia
InternetProvider
Retail
LogisticServices

HPC and graph analyzing algorithms

➡ Different parallel architectures

➡ Different programming models

➡ Different levels of fit to irregular problems & graph algorithms

Parallel Algorithms for network analysis
Challenges

Critical to consider graph size and topology in application to architecture mapping

- Can achieve high performance on GPUs if the graph + data structures fit in device memory (memory size wall).
- Reasonably good performance on distributed memory clusters if the graph has low conductance (can be partitioned w/ low edge cut).

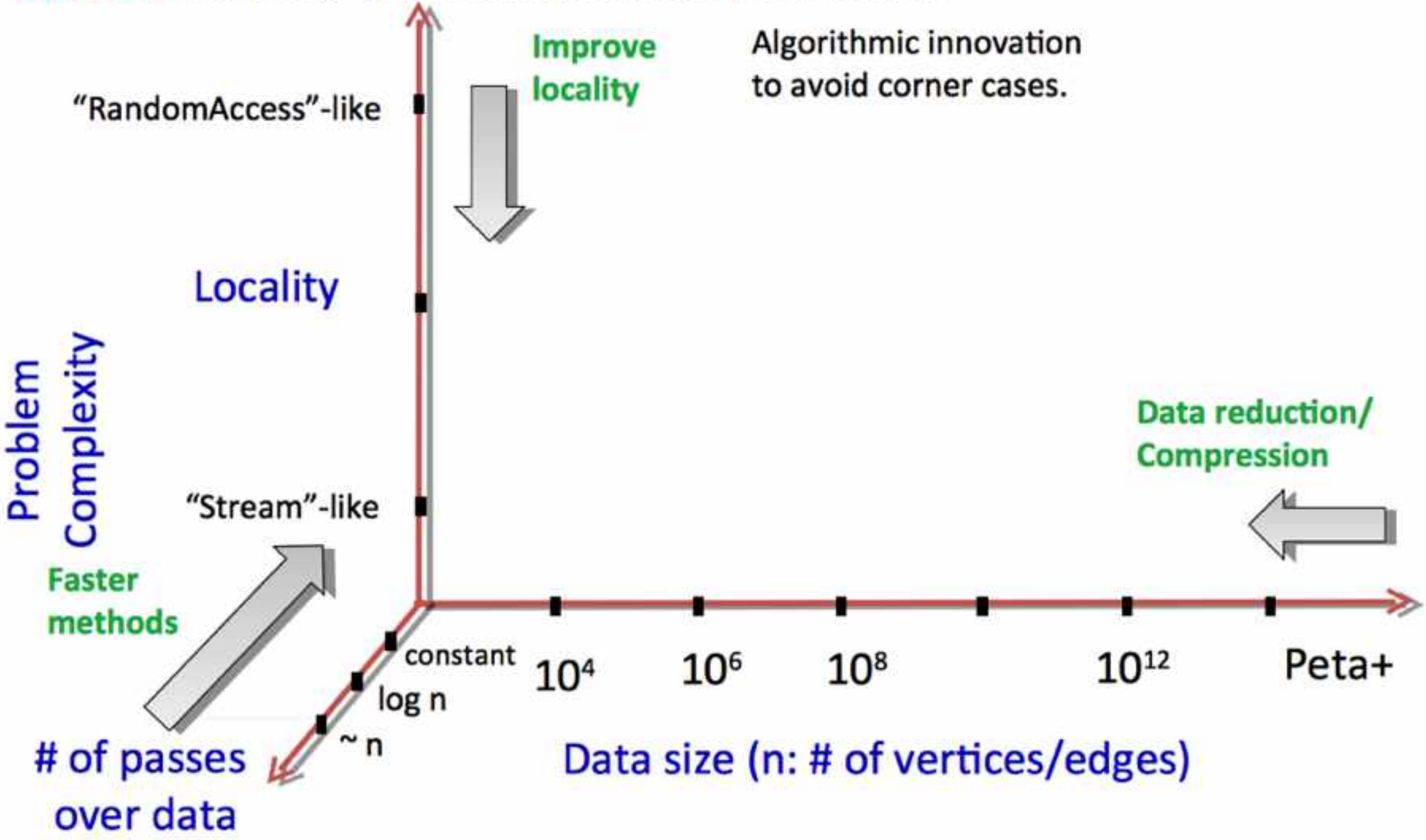
Latency: (memory wall) as most of memory is hundreds or thousands of cycles away from the processor that wants it.

- Computations that follow the edges of **irregular** graphs are unavoidably latency-limited (worst cases with poor locality)...

Designing fast parallel graph algorithms

System requirements: High (on-chip memory, DRAM, network, IO) bandwidth.

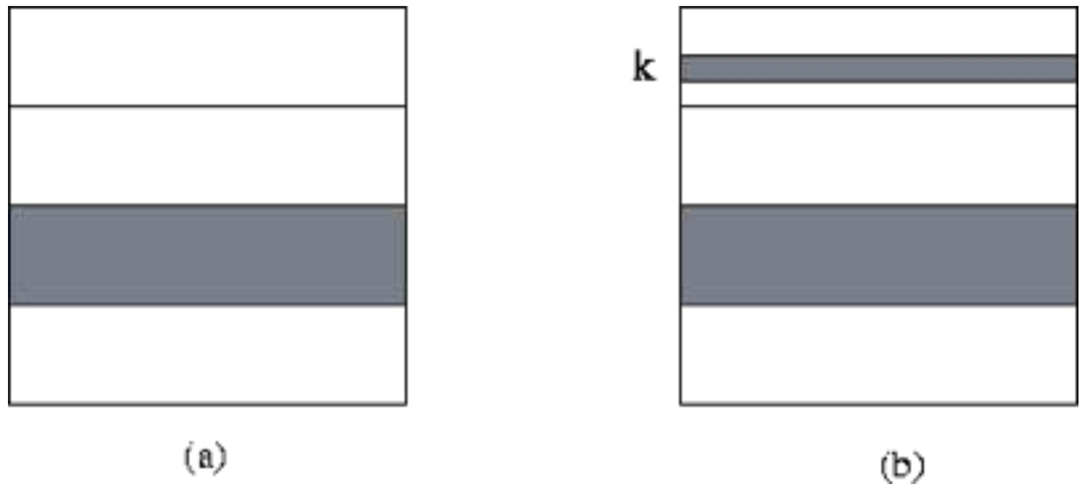
Solution: Efficiently utilize available memory bandwidth.



Example: Parallel “Shortest Paths”: Floyd–Warshall algorithm

$$D(0) = \begin{bmatrix} 0 & 3 & 8 & \infty & -4 \\ \infty & 0 & \infty & 1 & 7 \\ \infty & 4 & 0 & \infty & \infty \\ 2 & \infty & -5 & 0 & \infty \end{bmatrix} \Rightarrow D_{(k-1)} = \begin{bmatrix} 0 & 3 & 8 & \infty & -4 \\ \infty & 0 & \infty & 1 & 7 \\ \infty & 4 & 0 & \infty & \infty \\ 2 & 5 & -5 & 0 & -2 \\ \infty & \infty & \infty & 6 & 0 \end{bmatrix}$$

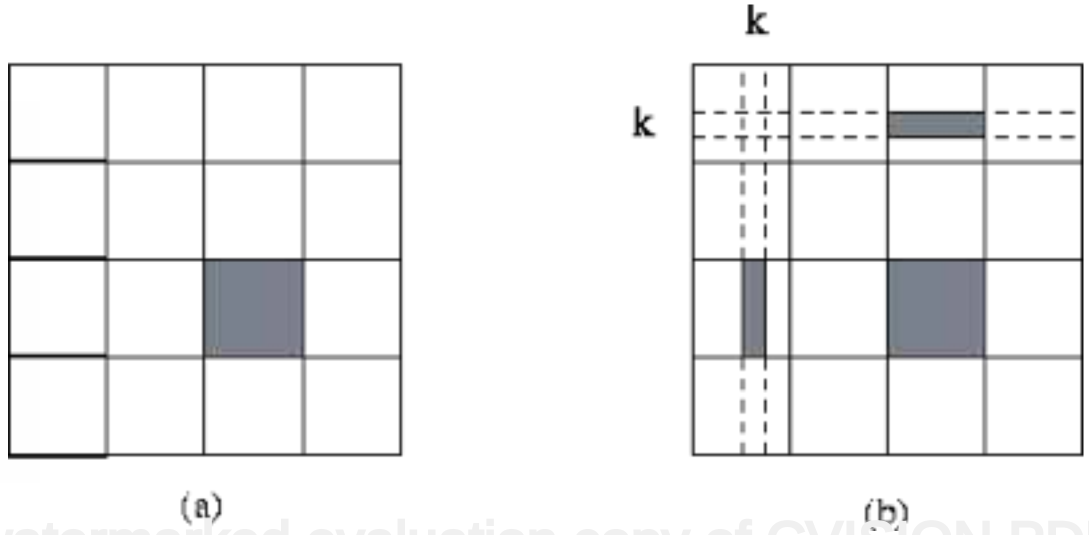
```
For k=1 to n {
  For i=1 to n {            Parallel
     For j=1 to n
        D[i,j] = min(D[i,j],D[i,k]+D[k,j])
  }
}
```



(a) (b)

Time > N^3/p

```
For k=1 to n {
  For i=1 to n {            Parallel
     For j=1 to n           Parallel
        D[i,j] = min(D[i,j],D[i,k]+D[k,j])
  }
}
```



(a) (b)

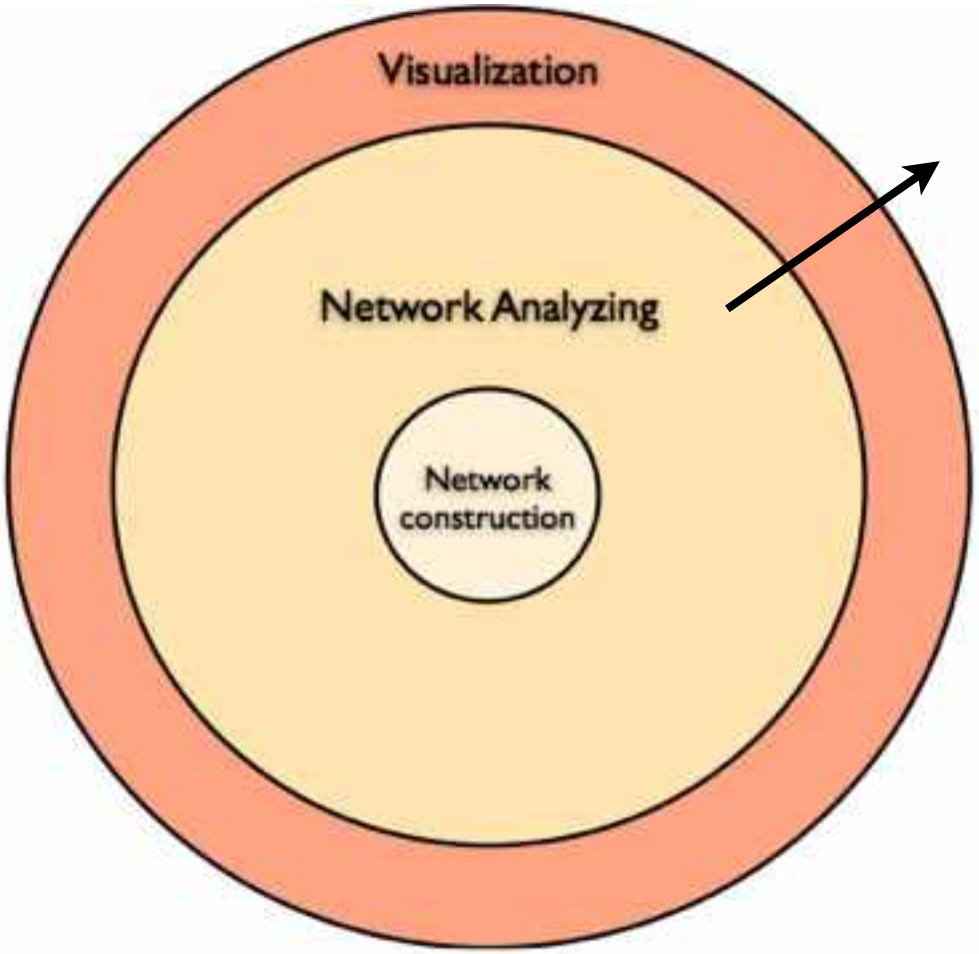
Time > N^3/p

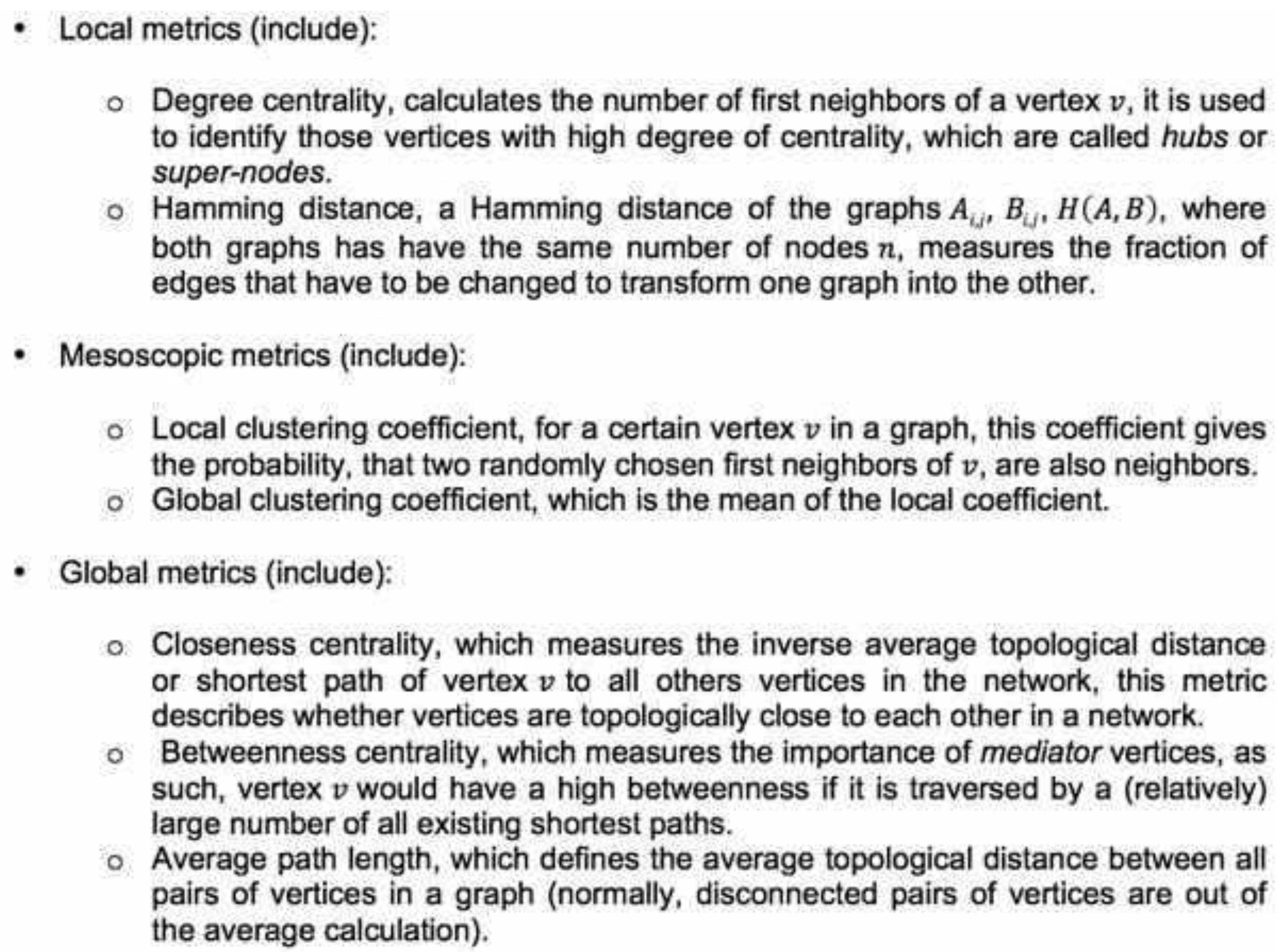
Outline

- Software Engineering
  - ➡ Importance
  - ➡ Process
- Network analysis
  - ➡ Data mining and Networks approach
  - ➡ Computational issues
  - ➡ Parallel network algorithms
- Software Production: parallel “Igraph” algorithms
  - ➡ How is it being developed: “SE”
  - ➡ What next?

Parallel Algorithms for network analysis
Requirements:

Fast and Correct





- Local metrics (include):
  - Degree centrality, calculates the number of first neighbors of a vertex v, it is used to identify those vertices with high degree of centrality, which are called *hubs* or *super-nodes*.
  - Hamming distance, a Hamming distance of the graphs $A_{i,j}$, $B_{i,j}$, $H(A,B)$, where both graphs has have the same number of nodes n, measures the fraction of edges that have to be changed to transform one graph into the other.
- Mesoscopic metrics (include):
  - Local clustering coefficient, for a certain vertex v in a graph, this coefficient gives the probability, that two randomly chosen first neighbors of v, are also neighbors.
  - Global clustering coefficient, which is the mean of the local coefficient.
- Global metrics (include):
  - Closeness centrality, which measures the inverse average topological distance or shortest path of vertex v to all others vertices in the network, this metric describes whether vertices are topologically close to each other in a network.
  - Betweenness centrality, which measures the importance of *mediator* vertices, as such, vertex v would have a high betweenness if it is traversed by a (relatively) large number of all existing shortest paths.
  - Average path length, which defines the average topological distance between all pairs of vertices in a graph (normally, disconnected pairs of vertices are out of the average calculation).

J.F. Donges, Y. Zou, N. Marwan and J. Kurths, "Complex Networks in Climate Dynamics", *Eur. Phys. J. Special Topics 174, 157-179 (2009).*

Parallel Algorithms for network analysis

Design

➡ Start from scratch

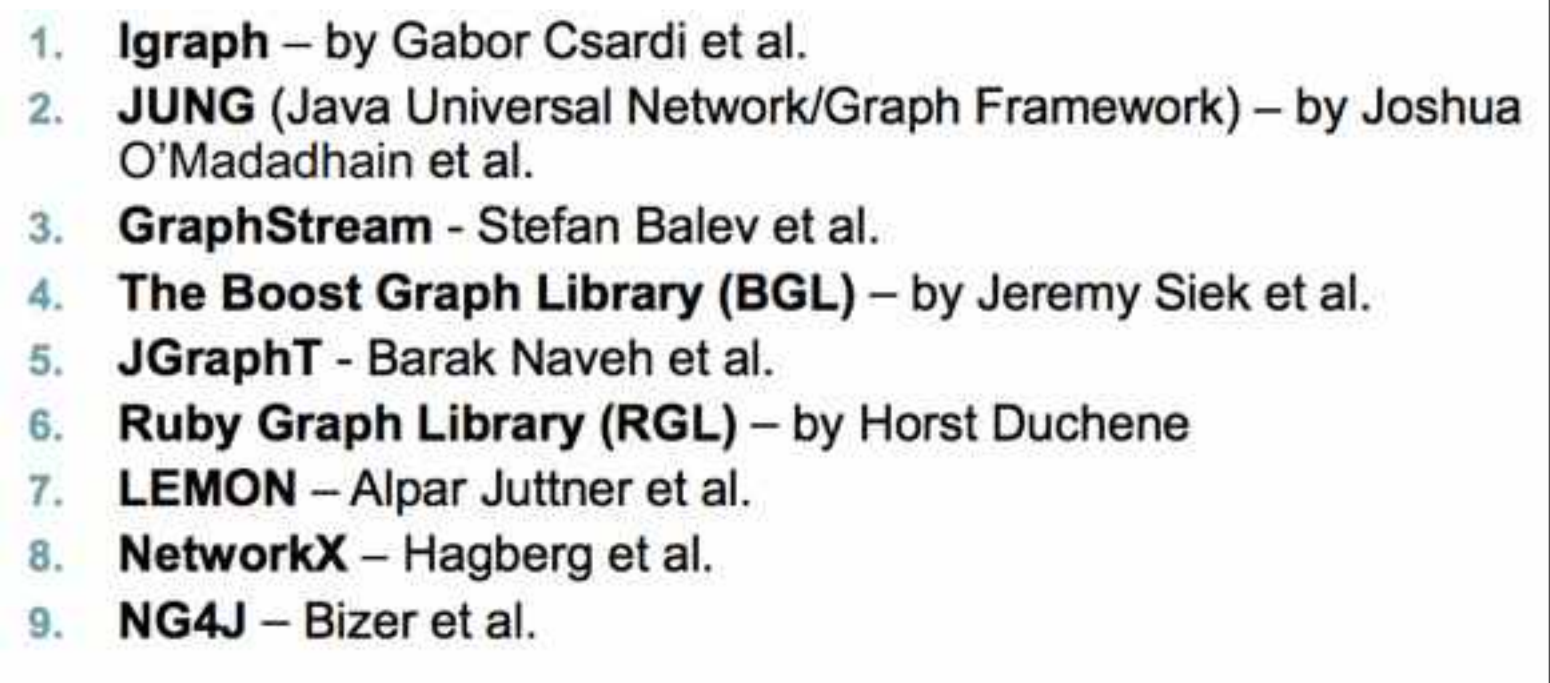
➡ Parallelize an existing library

Parallel Algorithms for network analysis

Design

➡ Start from scratch

➡ Parallelize an existing library



1. **Igraph** – by Gabor Csardi et al.
2. **JUNG** (Java Universal Network/Graph Framework) – by Joshua O'Madadhain et al.
3. **GraphStream** - Stefan Balev et al.
4. **The Boost Graph Library (BGL)** – by Jeremy Siek et al.
5. **JGraphT** - Barak Naveh et al.
6. **Ruby Graph Library (RGL)** – by Horst Duchene
7. **LEMON** – Alpar Juttner et al.
8. **NetworkX** – Hagberg et al.
9. **NG4J** – Bizer et al.

Parallel Algorithms for network analysis

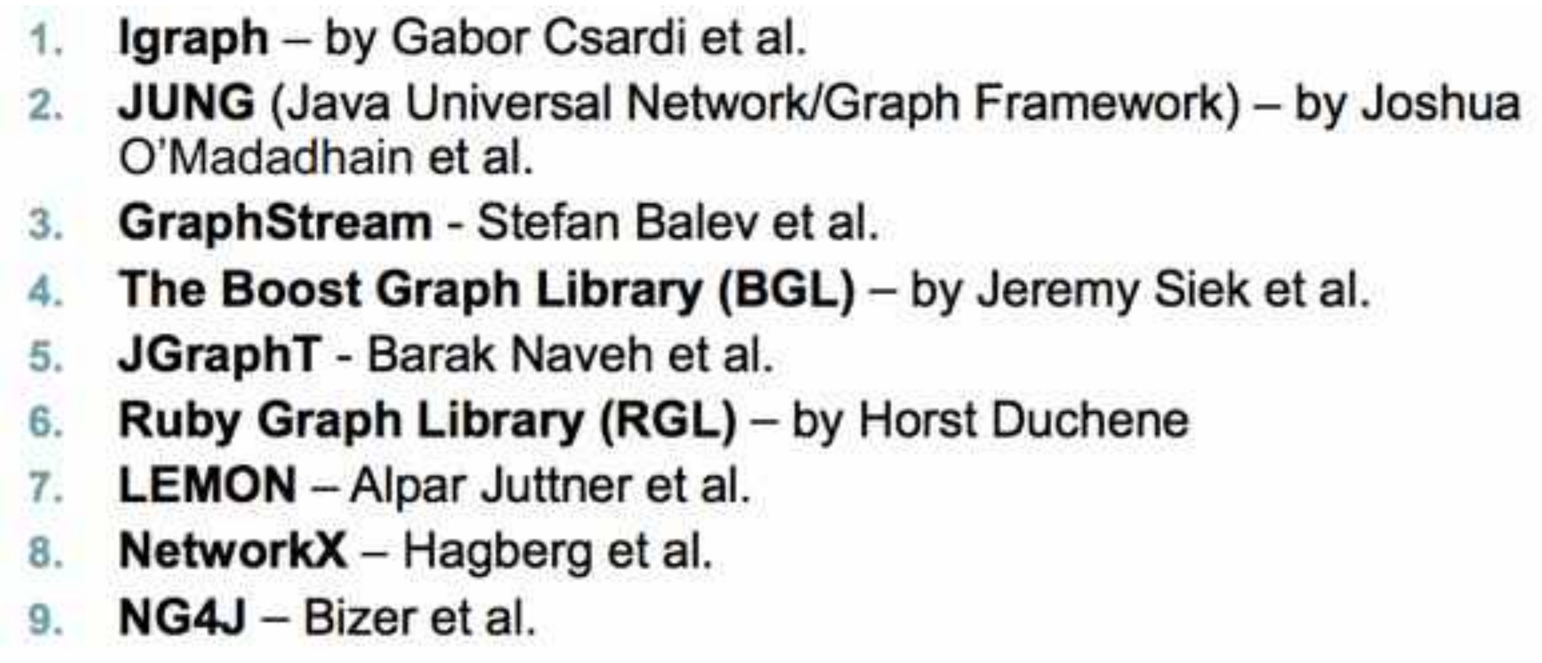
Design

➡ Start from scratch

➡ Parallelize an existing library

Selection Criteria:

Data Structure flexibility
Language
Implementation of alternative algorithms
Documentation
Acceptance
Reliability



1. **Igraph** – by Gabor Csardi et al.
2. **JUNG** (Java Universal Network/Graph Framework) – by Joshua O'Madadhain et al.
3. **GraphStream** - Stefan Balev et al.
4. **The Boost Graph Library (BGL)** – by Jeremy Siek et al.
5. **JGraphT** - Barak Naveh et al.
6. **Ruby Graph Library (RGL)** – by Horst Duchene
7. **LEMON** – Alpar Juttner et al.
8. **NetworkX** – Hagberg et al.
9. **NG4J** – Bizer et al.

Parallel Algorithms for network analysis

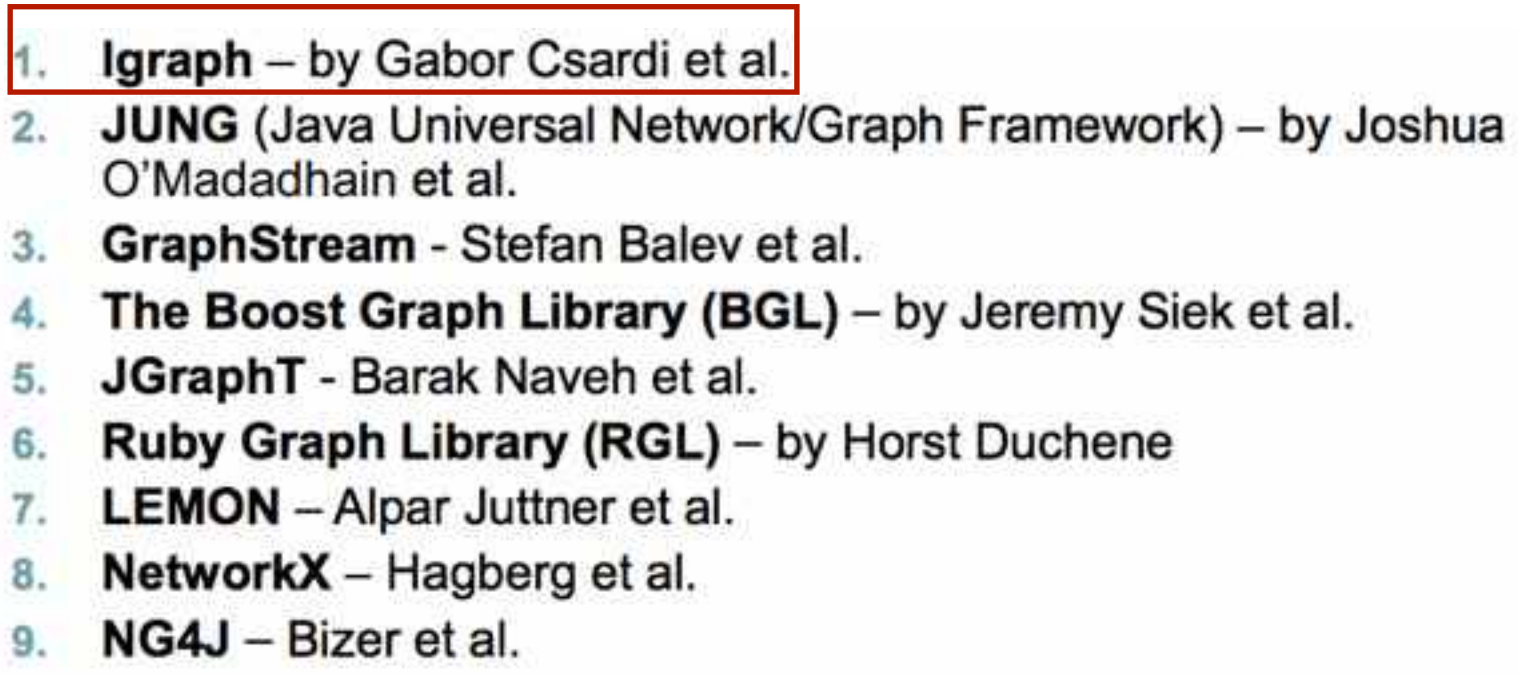
Design

➡ Start from scratch

➡ Parallelize an existing library

Selection Criteria:

Data Structure flexibility
Language
Implementation of alternative algorithms
Documentation
Acceptance
Reliability

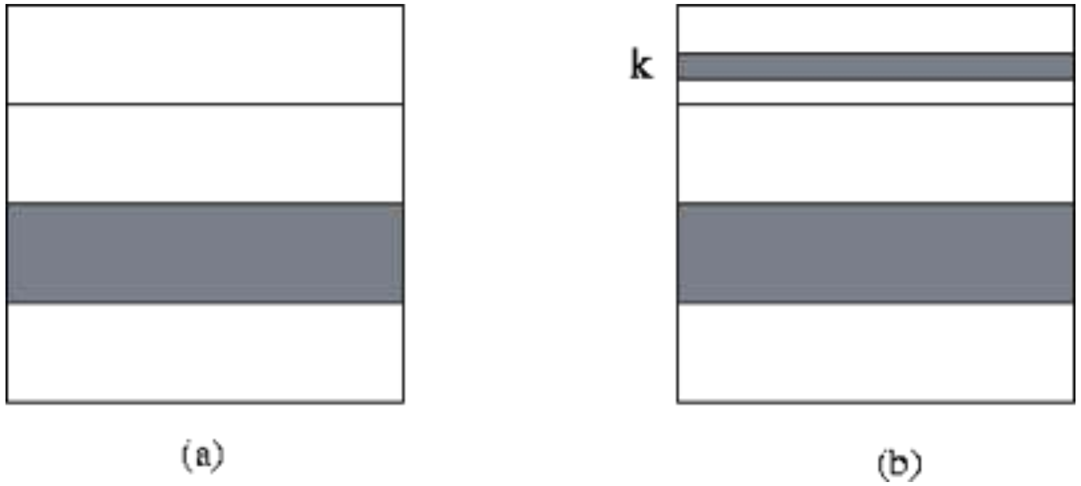


1. **Igraph** – by Gabor Csardi et al.
2. **JUNG** (Java Universal Network/Graph Framework) – by Joshua O'Madadhain et al.
3. **GraphStream** - Stefan Balev et al.
4. **The Boost Graph Library (BGL)** – by Jeremy Siek et al.
5. **JGraphT** - Barak Naveh et al.
6. **Ruby Graph Library (RGL)** – by Horst Duchene
7. **LEMON** – Alpar Juttner et al.
8. **NetworkX** – Hagberg et al.
9. **NG4J** – Bizer et al.

Design: Parallel "Shortest Paths": Floyd–Warshall algorithm

$$D(0) = \begin{bmatrix} 0 & 3 & 8 & \infty & -4 \\ \infty & 0 & \infty & 1 & 7 \\ \infty & 4 & 0 & \infty & \infty \\ 2 & \infty & -5 & 0 & \infty \end{bmatrix} \Rightarrow D_{(k-1)} = \begin{bmatrix} 0 & 3 & 8 & \infty & -4 \\ \infty & 0 & \infty & 1 & 7 \\ \infty & 4 & 0 & \infty & \infty \\ 2 & 5 & -5 & 0 & -2 \\ \infty & \infty & \infty & 6 & 0 \end{bmatrix}$$

```
For k=1 to n {
  For i=1 to n {            Parallel
     For j=1 to n
        D[i,j] = min(D[i,j],D[i,k]+D[k,j])
  }
}
```

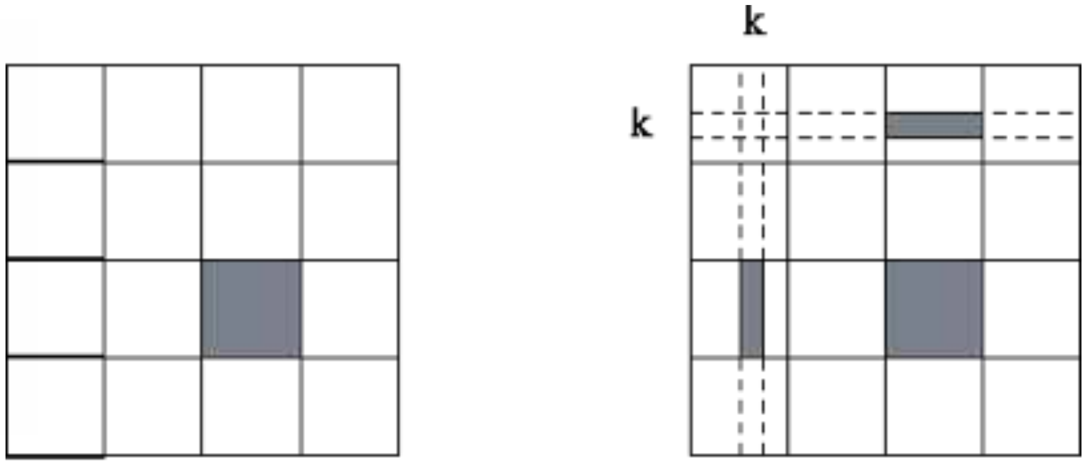


(a)

(b)

Time > N^3/p

```
For k=1 to n {
  For i=1 to n {          Parallel
     For j=1 to n          Parallel
        D[i,j] = min(D[i,j],D[i,k]+D[k,j])
  }
}
```



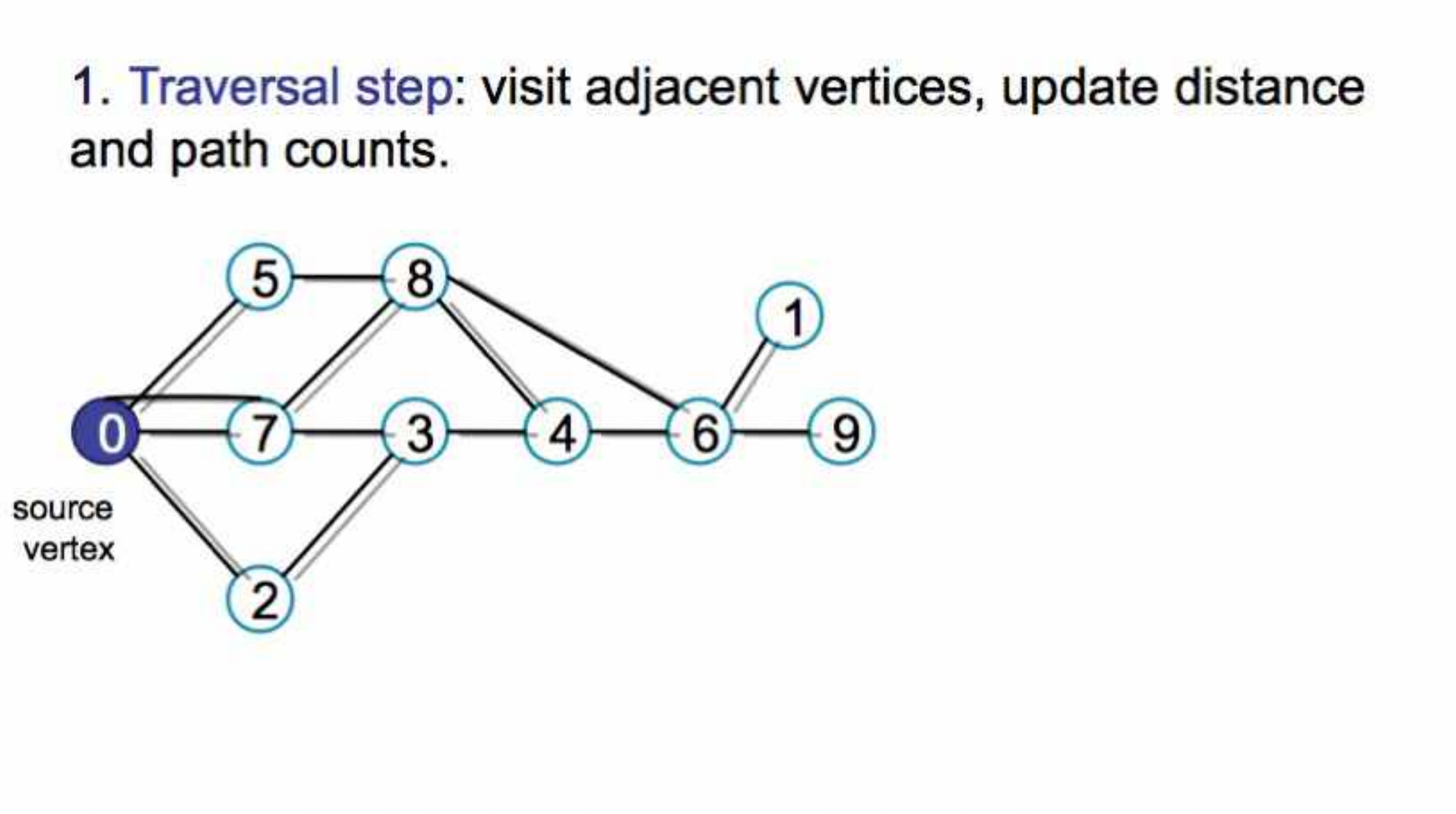
Time > N^3/p

Parallel Algorithms for network analysis

Design

Parallel Design BC Algorithm Illustration

SNAP: Small-world Network Analysis and Partitioning (Madduri and Bader 2009)



1. Traversal step: visit adjacent vertices, update distance and path counts.

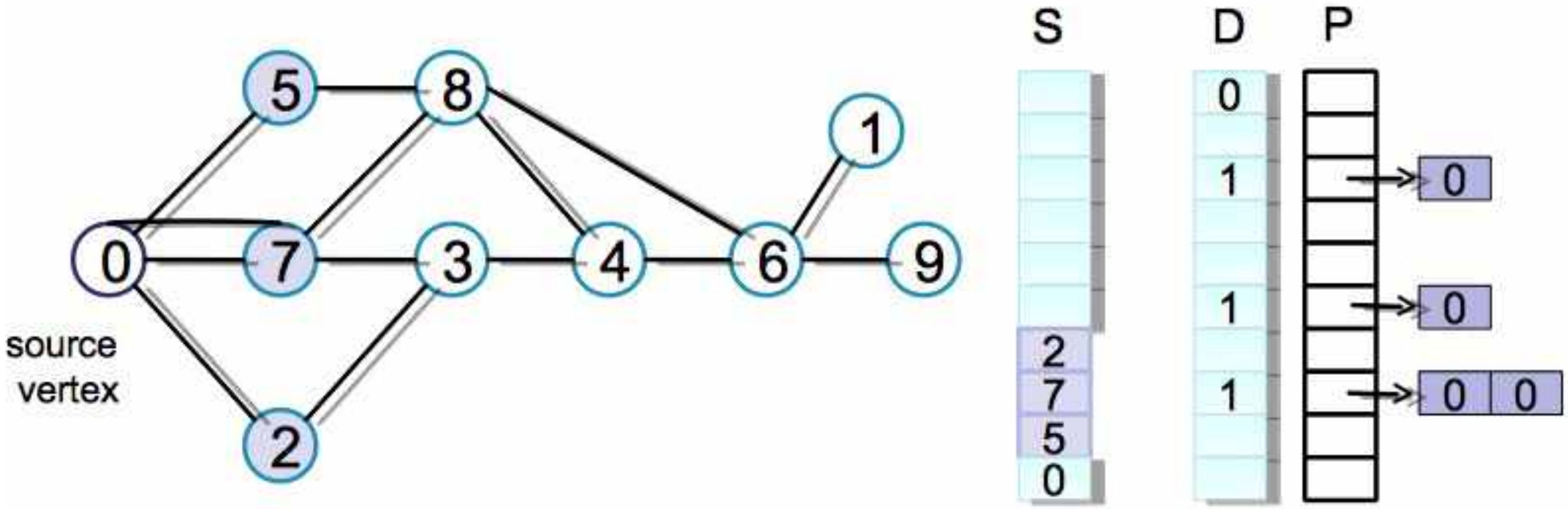
Parallel Algorithms for network analysis

Design

Parallel Design BC Algorithm Illustration

SNAP: Small-world Network
Analysis and Partitioning (Madduri and Bader 2009)

1. Traversal step: visit adjacent vertices, update distance and path counts.

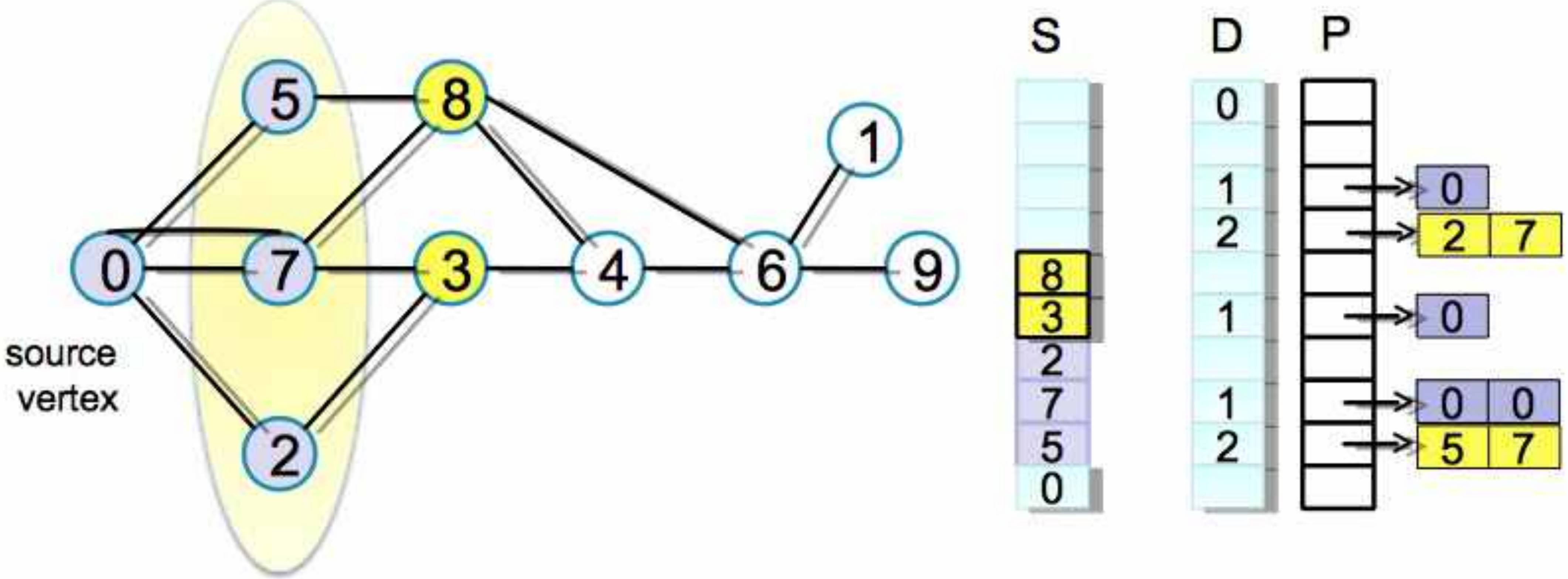


Parallel Algorithms for network analysis
Design

Parallel Design BC Algorithm Illustration

SNAP: Small-world Network
Analysis and Partitioning (Madduri and Bader 2009)

1. Traversal step: visit adjacent vertices, update distance and path counts.



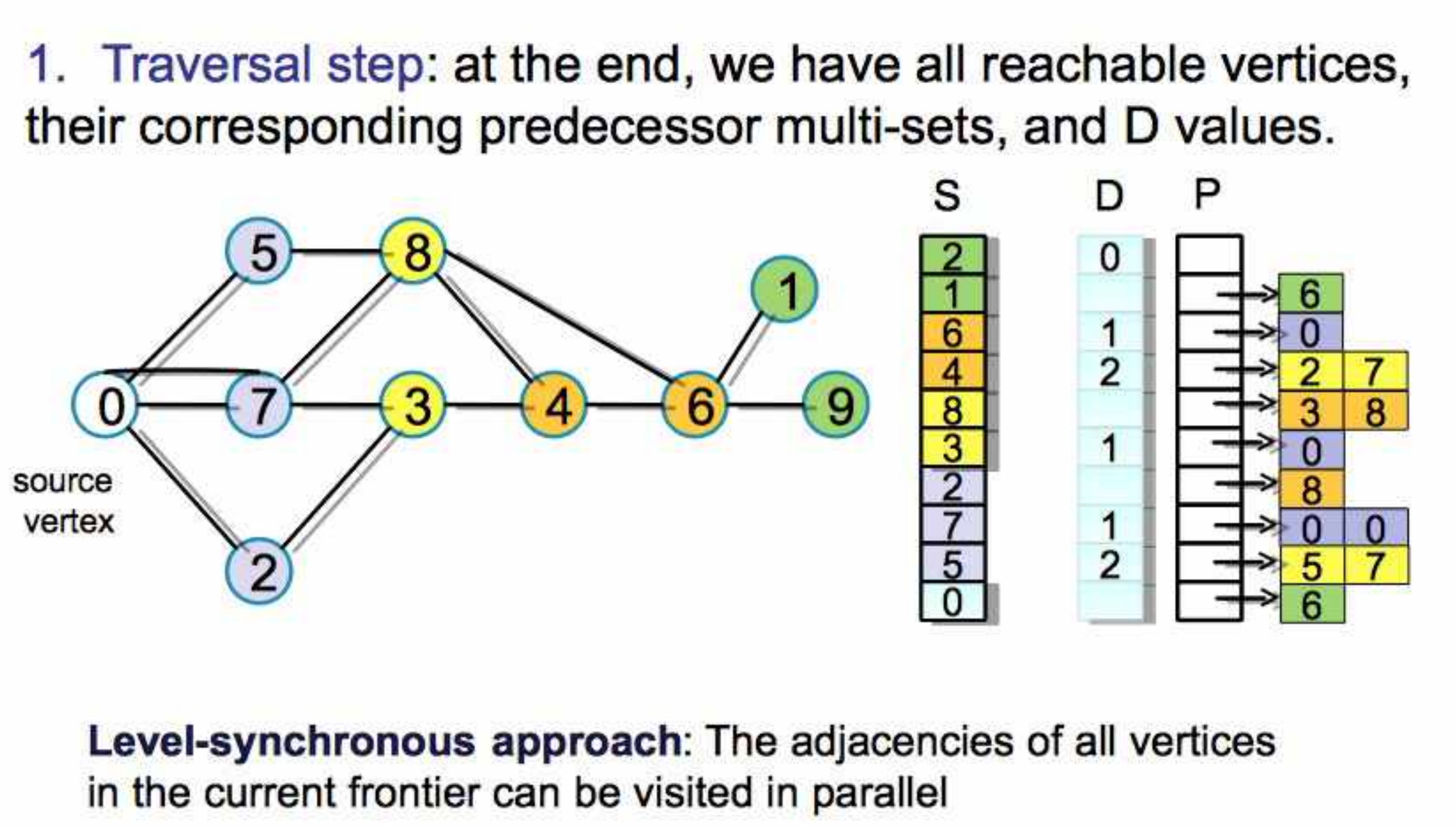
Level-synchronous approach: The adjacencies of all vertices in the current frontier can be visited in parallel

Parallel Algorithms for network analysis

Design

Parallel Design BC Algorithm Illustration

SNAP: Small-world Network
Analysis and Partitioning (Madduri and Bader 2009)



1. Traversal step: at the end, we have all reachable vertices, their corresponding predecessor multi-sets, and D values.

S	D	P
2	0	
1		6
6	1	0
4	2	2 7
8		3 8
3	1	0
2		8
7	1	0 0
5	2	5 7
0		6

Level-synchronous approach: The adjacencies of all vertices in the current frontier can be visited in parallel

Parallel Algorithms for network analysis
Design

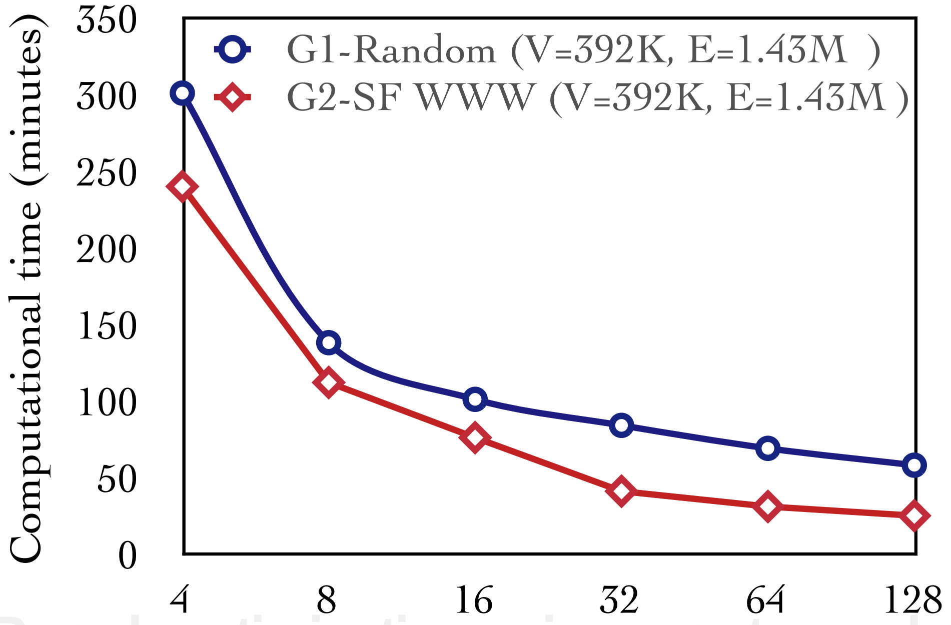
- **Implementation**
  - C -- OpenMP "threads"
  - Incremental + Versioning (igraph-ori -->v0.1, v0.2)

Parallel Algorithms for network analysis
Design

- Implementation
  - C -- OpenMP “threads”
  - Incremental + Versioning (igraph-ori -->v0.1, v0.2)

- Testing

Parallel “iGraph” B.C.



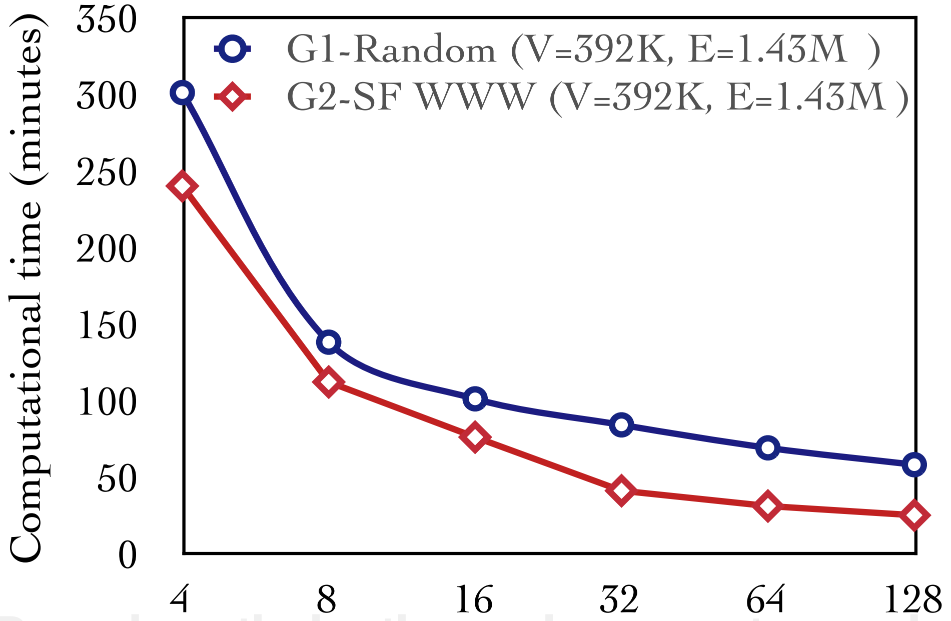
Parallel Algorithms for network analysis

Design

- **Implementation**
  - C -- OpenMP "threads"
  - Incremental + Versioning (igraph-ori -->v0.1, v0.2)

- **Testing**

Parallel "iGraph" B.C.



Documentation

What next?

- **Network Construction**
  - network size?
  - time?

What next?

- **Network Construction**
  - network size?
  - time?

- **More algorithms**

What next?

- Network Construction
  - network size?
  - time?

- Testing....

- More algorithms

Documentation