A Multilevel Design for Scalable Pareto-Optimal Public Transit Queries

Patrick Steil, November 26. 2025

Motivation

- demand for **large-scale** public transit routing

Motivation

- demand for **large-scale** public transit routing
- RAPTOR, CSA, Transfer Patterns, . . . struggle
  - either too slow, **or** require heavy preprocessing and memory
- Pareto-Optimal (arrival time, number of transfers)

Network Defintions

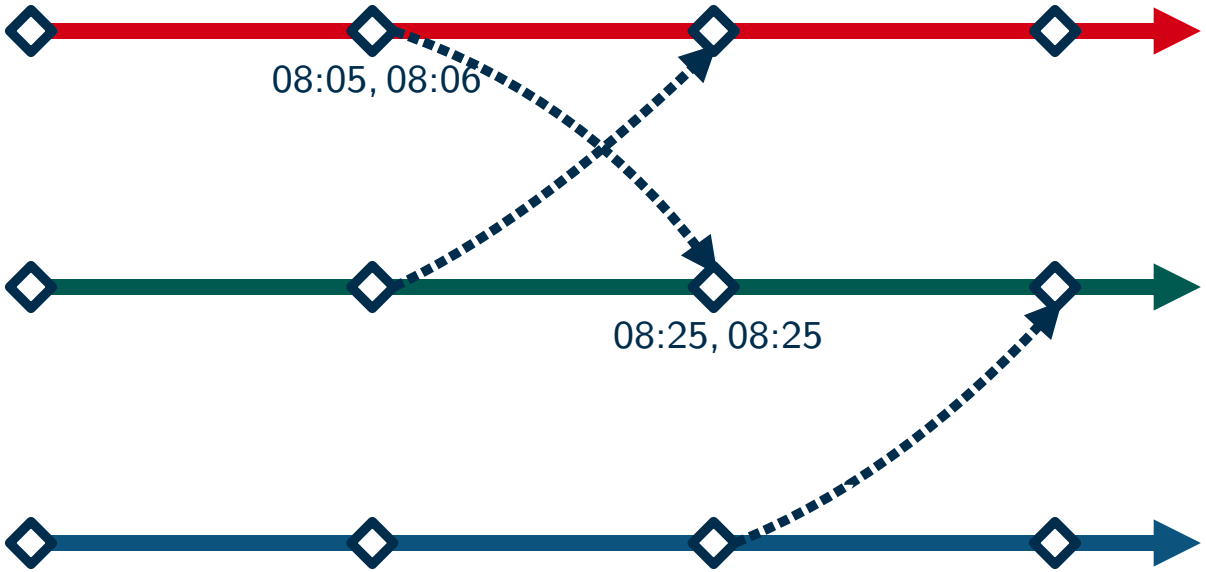
- Stops / Stations
- Lines / Routes
- Footpaths (between stops)

Network Defintions

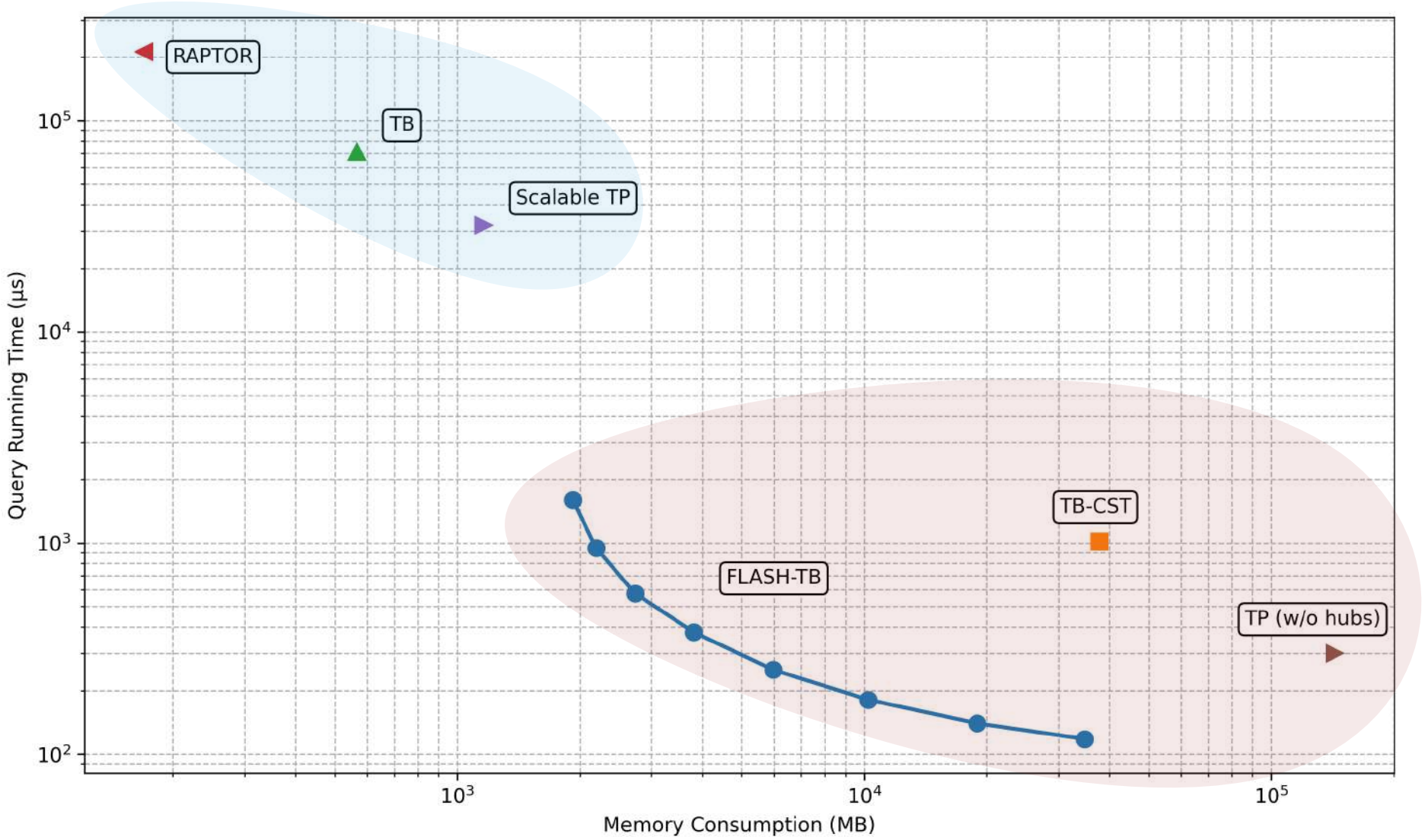
- Stops / Stations
- Lines
- Footpaths (between stops)
- Trips
- Stop Event (Arrival + Departure Event)

Network Definitions

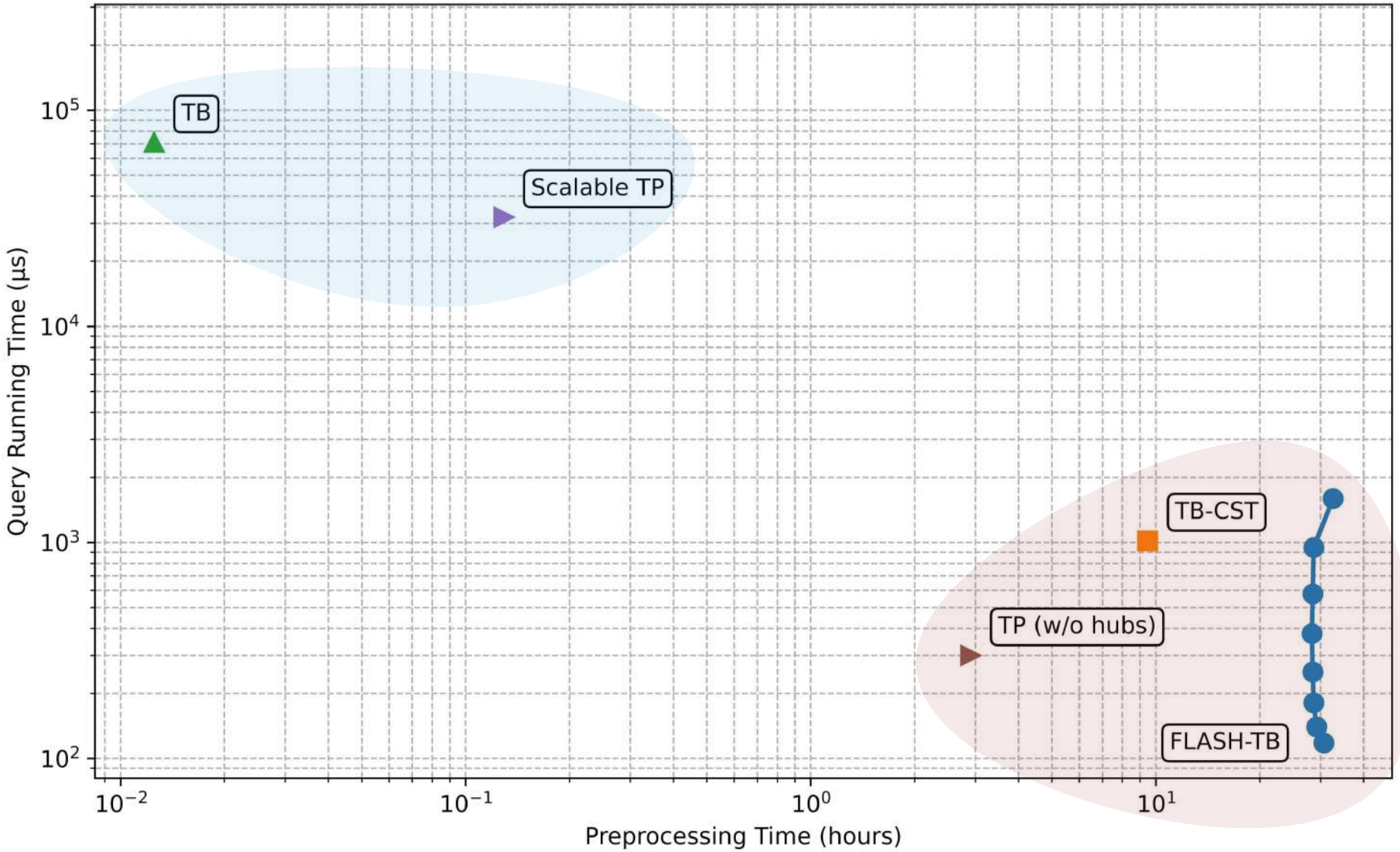
- Stops / Stations
- Lines
- Footpaths (between stops)
- Trips
- Stop Event (Arrival + Departure Event)
- Transfers



Trade-Off - Germany

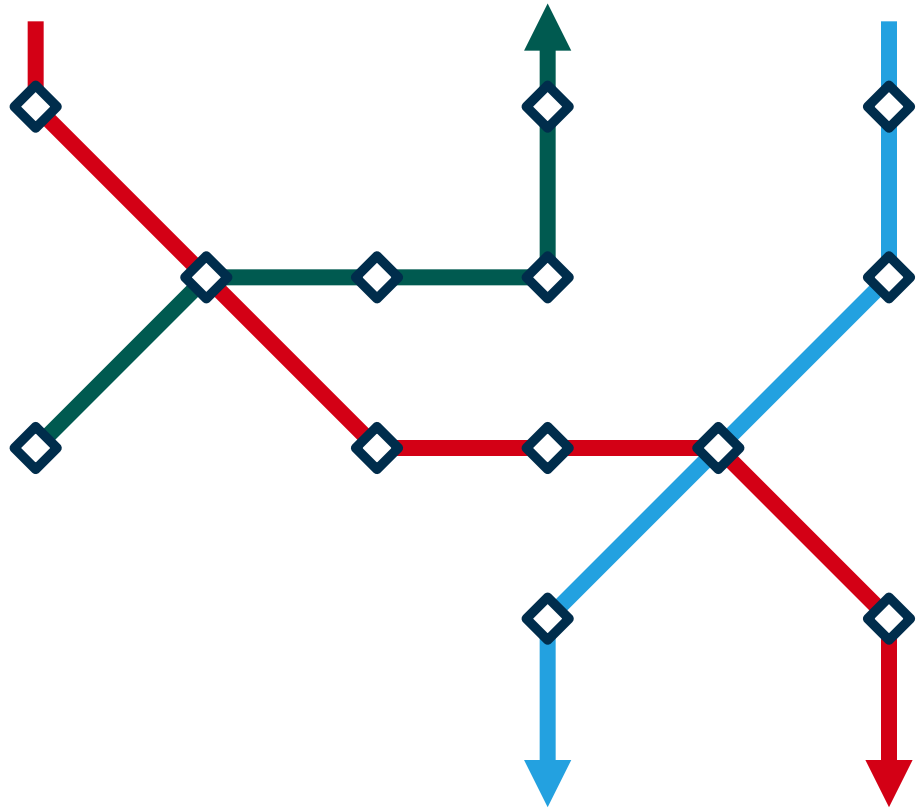


Trade-Off - Germany



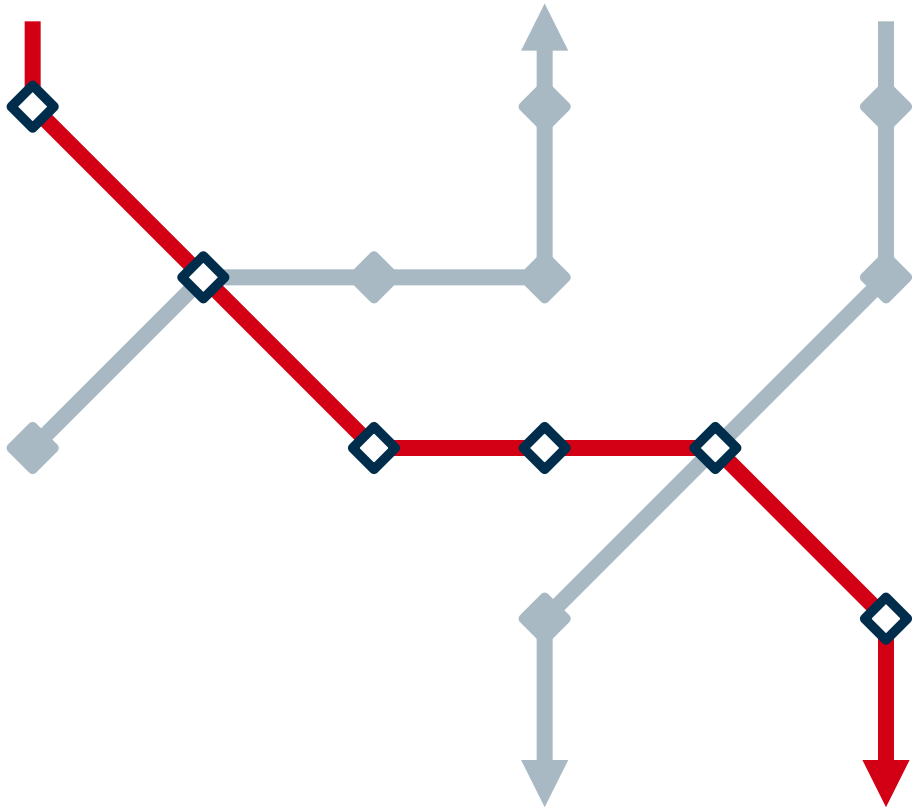
Trip-Based Routing

- BFS-like search
- for round $i = 0, 1, 2, \ldots$
  - scan collected trips
  - update arrival times
  - collect new trips via transfers



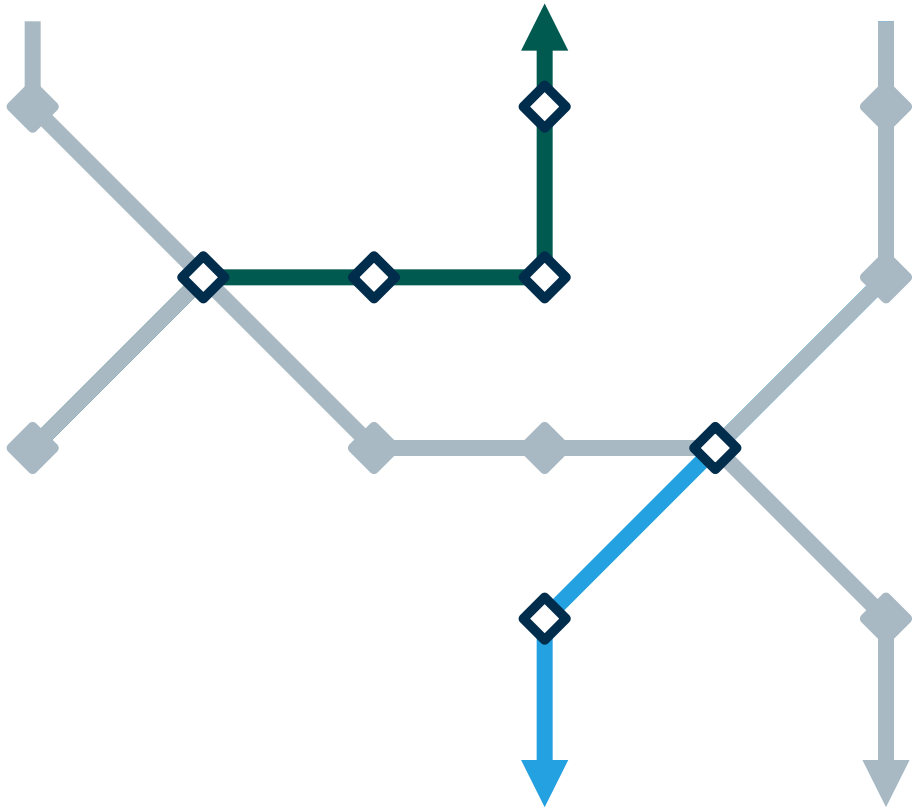
Trip-Based Routing

- BFS-like search
- for round $i = 0, 1, 2, \ldots$
  - scan collected trips
  - update arrival times
  - collect new trips via transfers



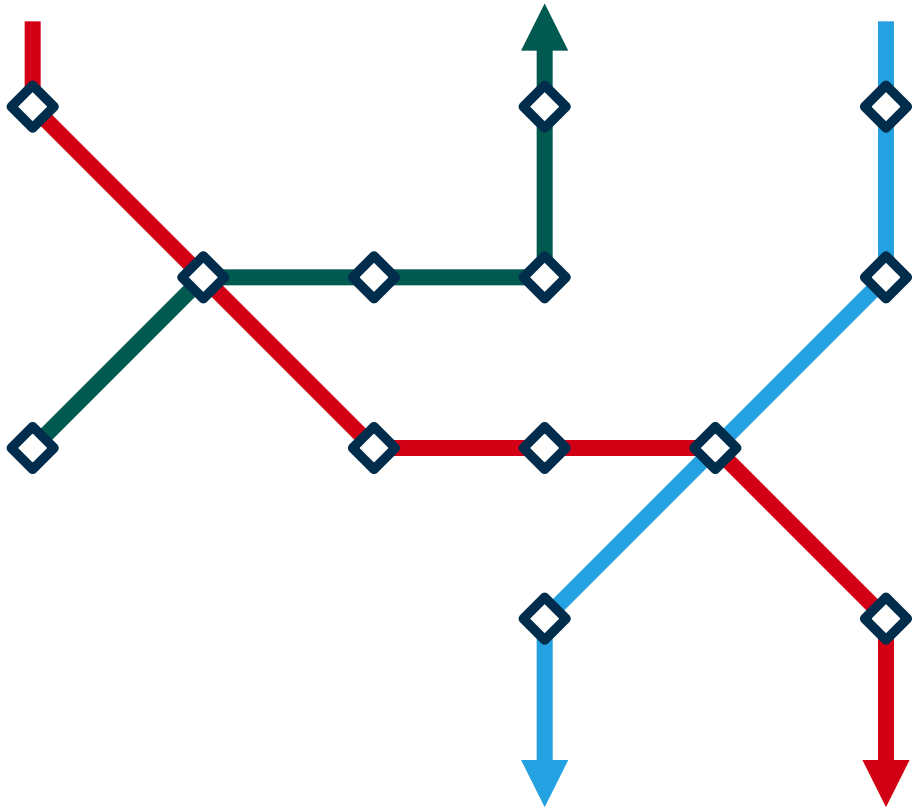
Trip-Based Routing

- BFS-like search
- for round $i = 0, 1, 2, \ldots$
  - scan collected trips
  - update arrival times
  - collect new trips via transfers



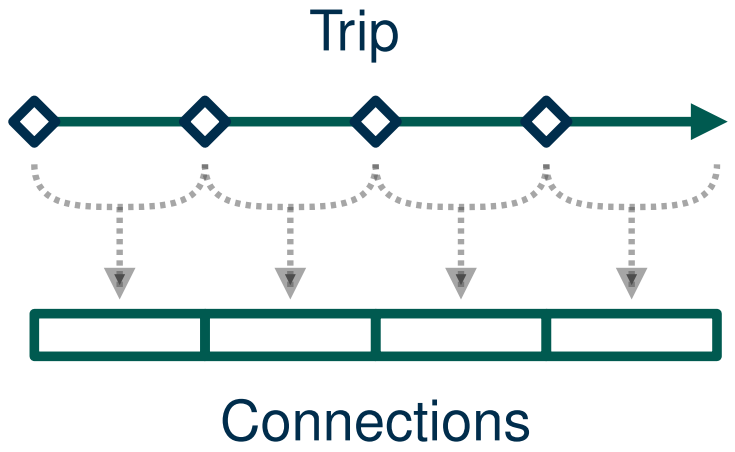
Trip-Based Routing

- BFS-like search
- for round $i = 0, 1, 2, \ldots$
  - scan collected trips
  - update arrival times
  - collect new trips via transfers
- small preprocessing (transfers)
- query too slow for large networks



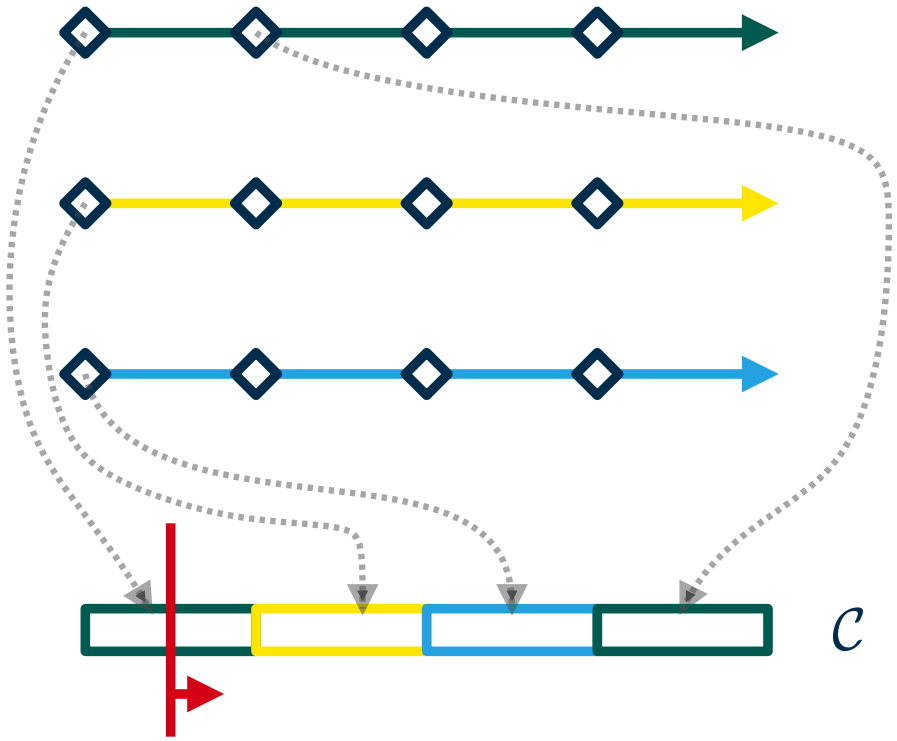
Connection Scan Algorithm

- only computes fastest journey
- based on connections



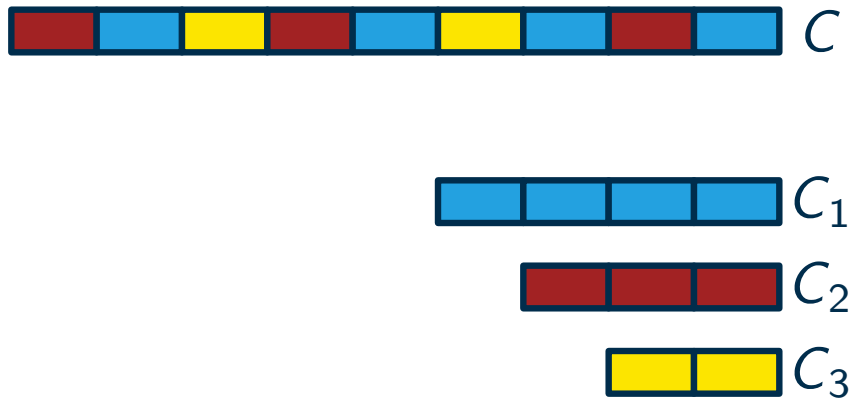
Connection Scan Algorithm

- only computes fastest journey
- based on connections
- maintain a sorted connections array $\mathcal{C}$
- linear sweep over the array like a dynamic program



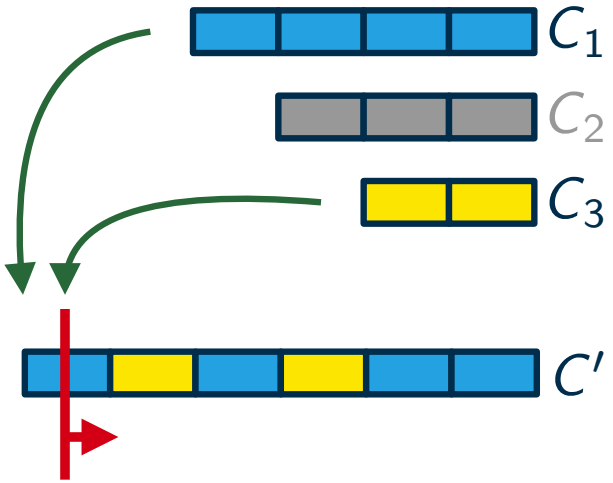
ACSA

- many unnecessary connections are scanned
- maintain many, smaller connections arrays $C_1, C_2, \ldots$



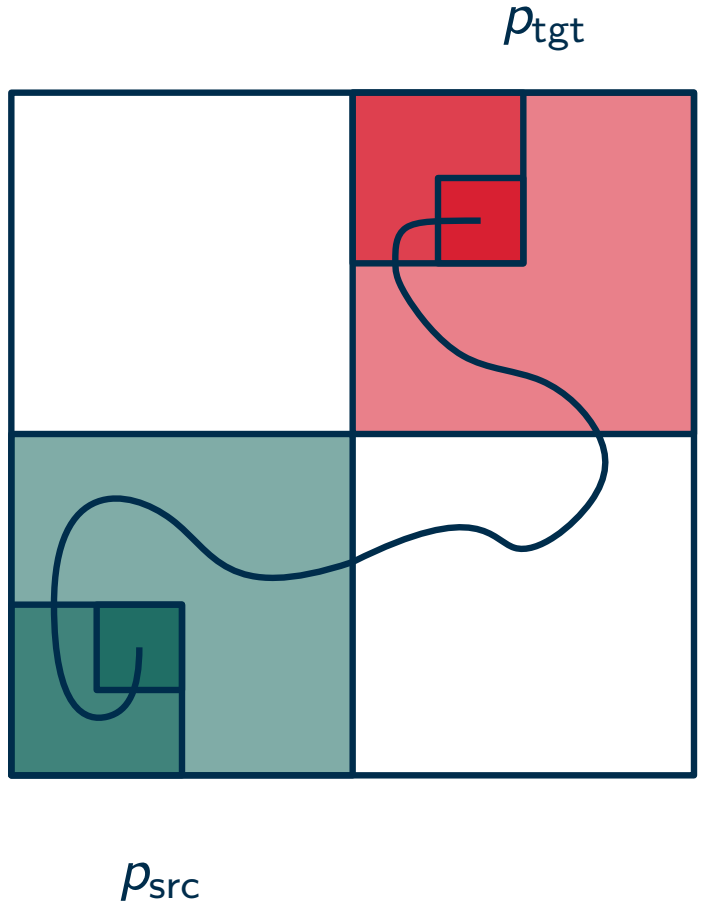
ACSA

- many unnecessary connections are scanned
- maintain many, smaller connections arrays $C_1, C_2, \ldots$
- Query:
  - merge only needed smaller connections arrays into C'
  - sort C' and run CSA



ACSA

- many unnecessary connections are scanned
- maintain many, smaller connections arrays $C_1, C_2, \ldots$
- Query:
  - merge only needed smaller connections arrays into C
  - sort C and run CSA
- successful multilevel approach
- drawbacks:
  - only non-Pareto journeys
  - *merging* phase dominates query
  - worse performance on metropolitain networks

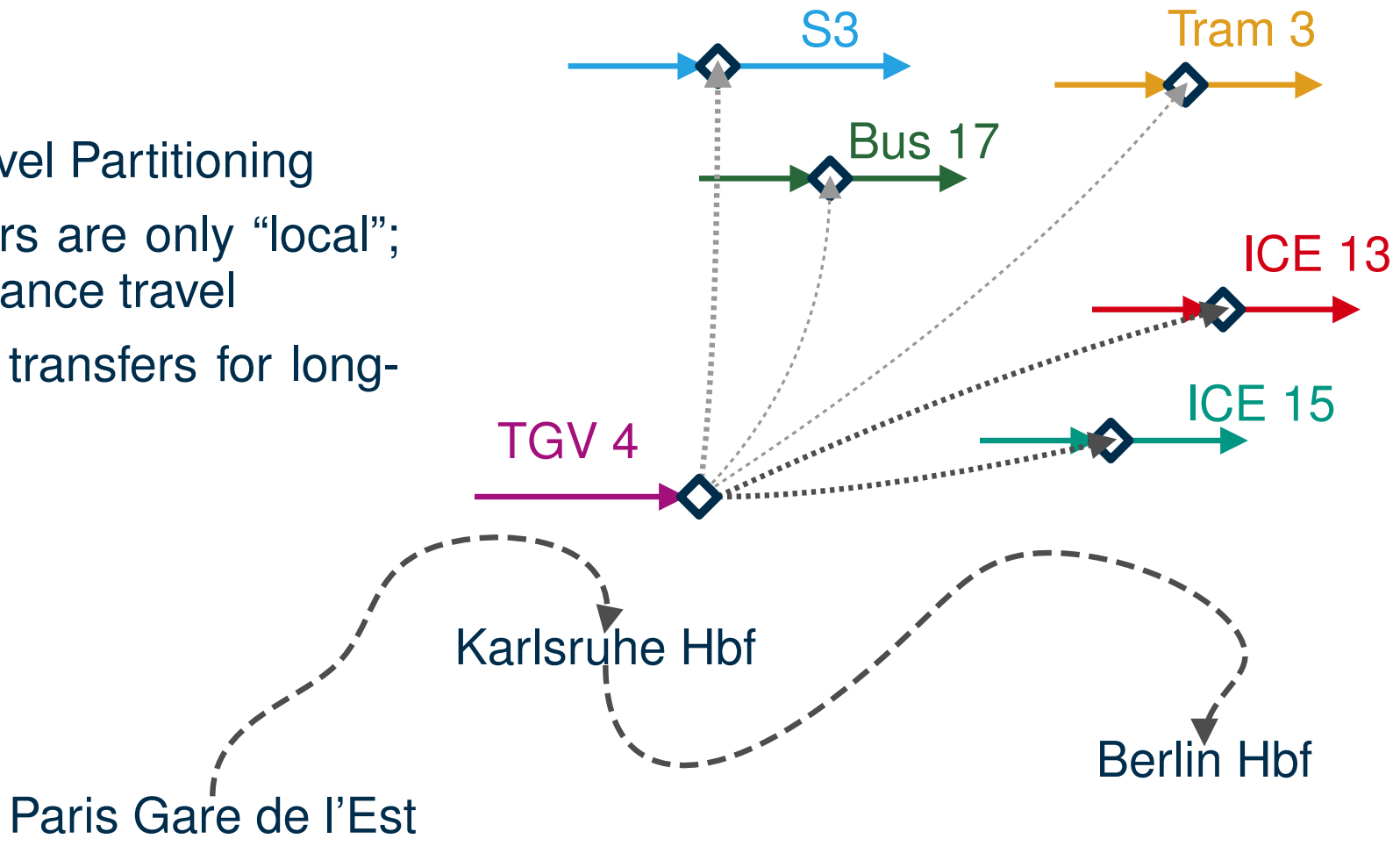


T-REX: Transfer-Ranked EXploration

- Combines TB with Multilevel Partitioning
- key insight: some transfers are only “local”; not important for long-distance travel

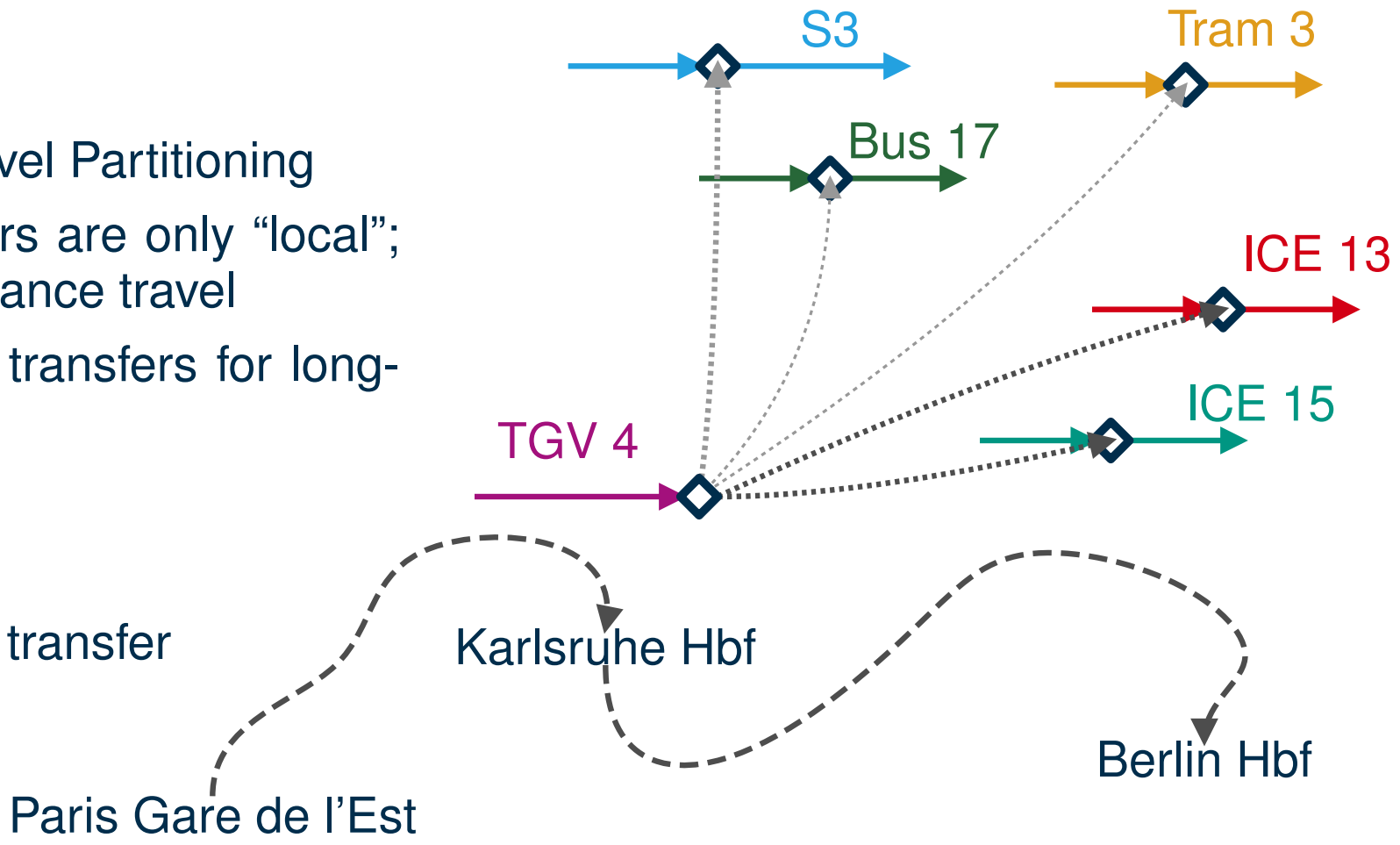
T-REX: Transfer-Ranked EXploration

- Combines TB with Multilevel Partitioning
- key insight: some transfers are only “local”; not important for long-distance travel
  - query can *ignore* local transfers for long-distance queries
- 3 main phases



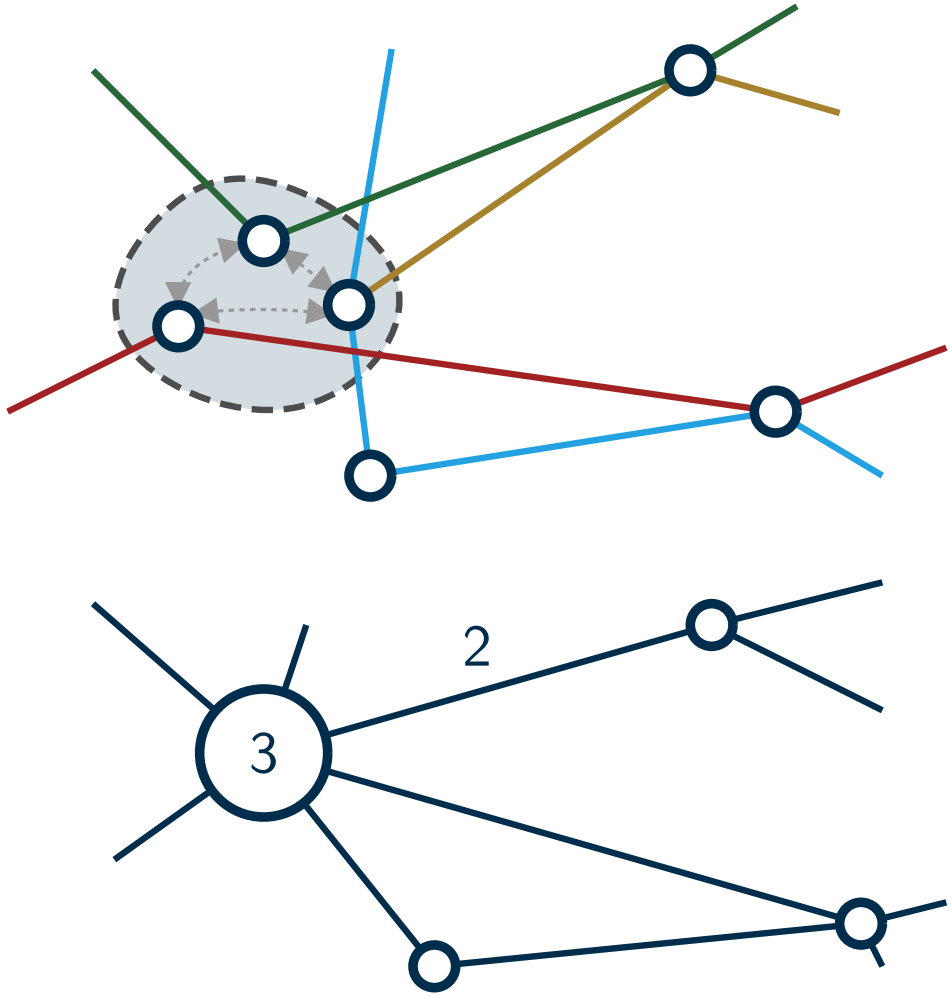
T-REX: Transfer-Ranked EXploration

- Combines TB with Multilevel Partitioning
- key insight: some transfers are only “local”; not important for long-distance travel
  - query can *ignore* local transfers for long-distance queries
- 3 main phases
  - Partition stops
  - Compute *rank* of every transfer
  - Query

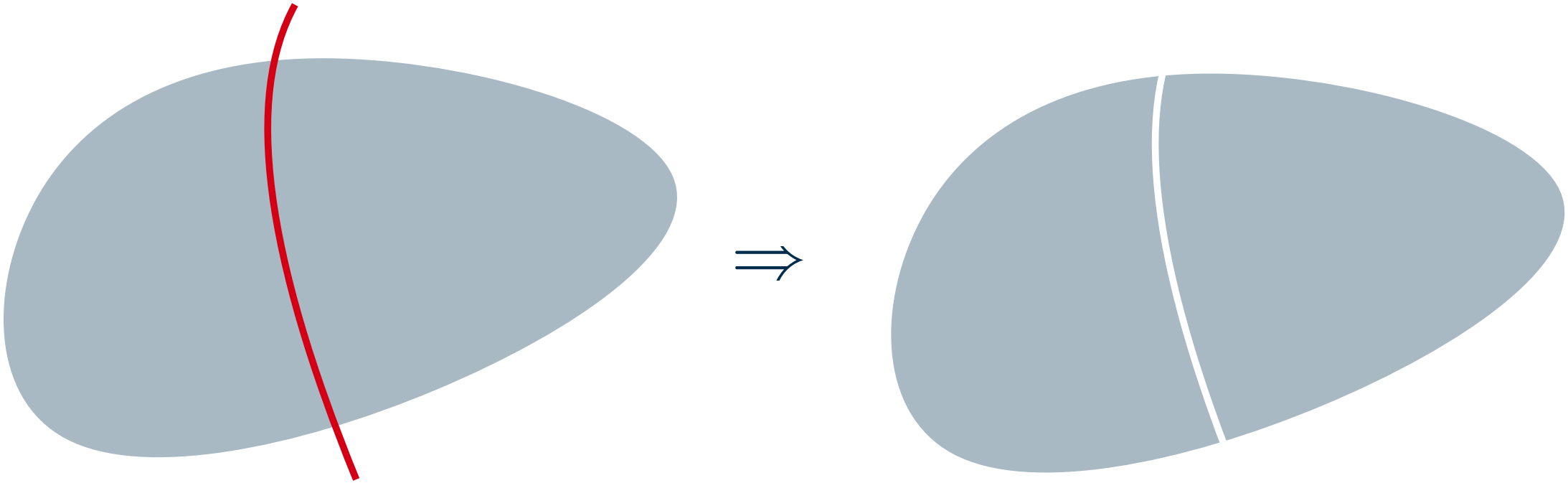


Partition Stops

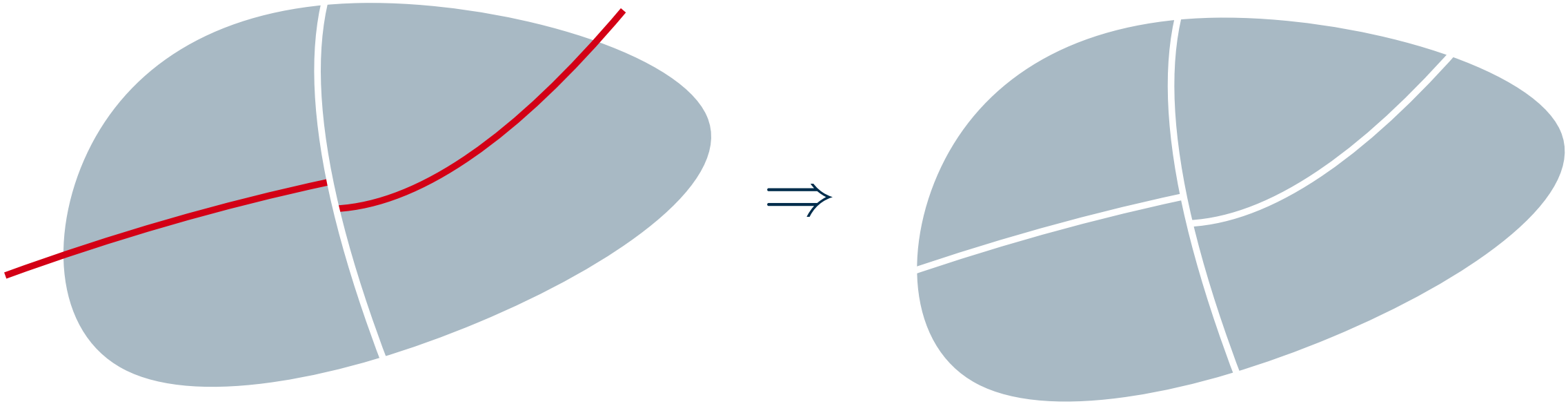
- Represent “connectivity” as graph
  - nodes represent stops
  - edges represent connections
- contract footpaths
- weight nodes and edges



Partition Stops



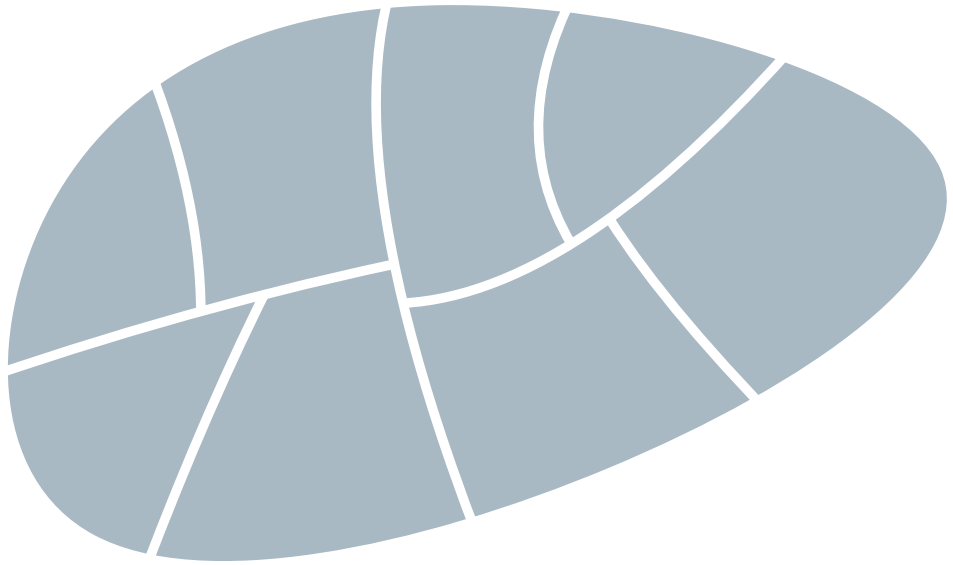
Partition Stops



Partition Stops

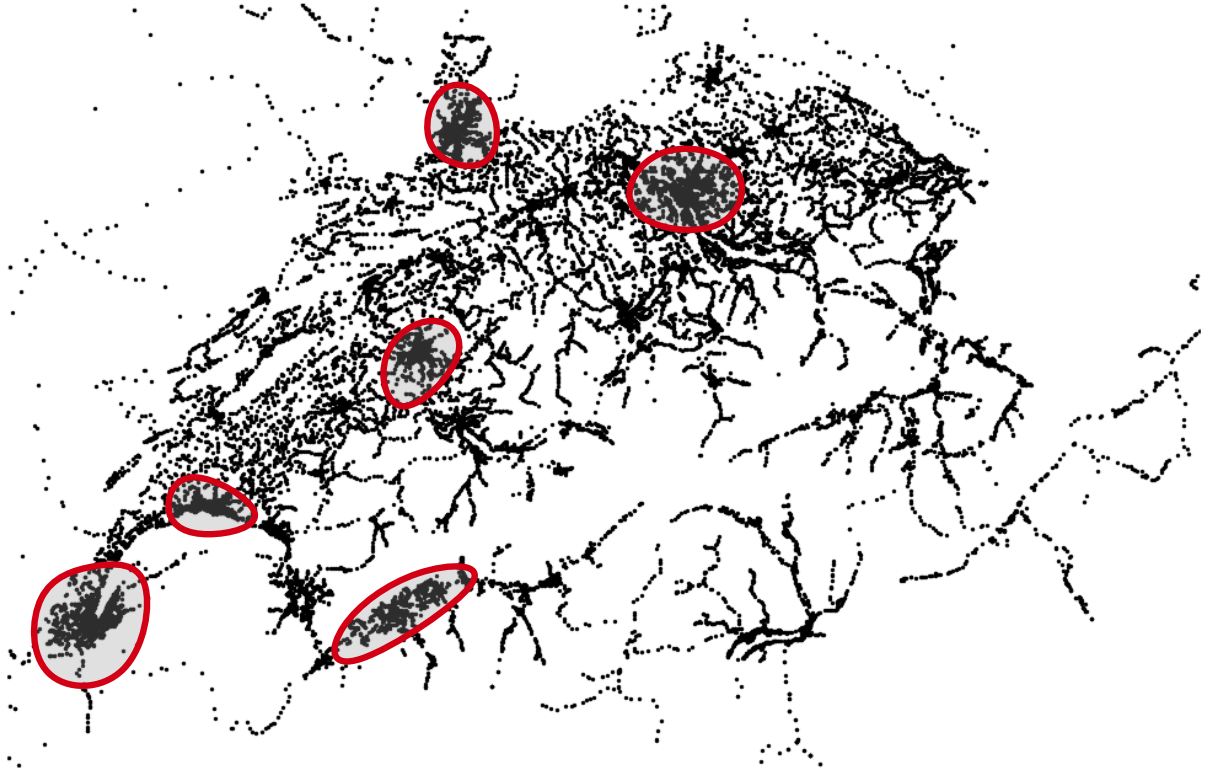
- partition nodes into 2^ℓ cells

$\ell = 3$



Partition Stops

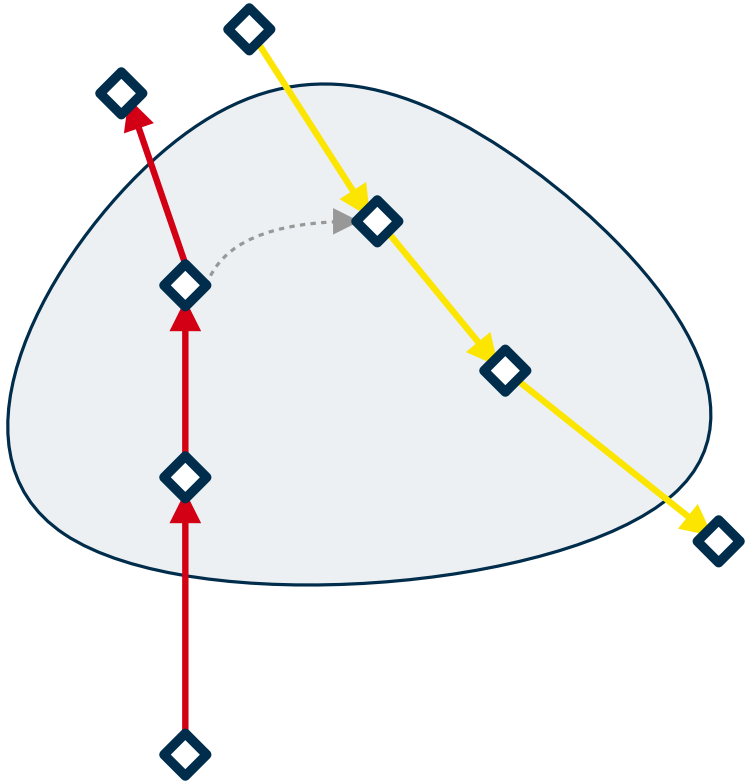
- partition nodes into 2^ℓ cells
- social graph-like-structures
  - metropolitan areas are densely connected
  - long-distance trains / busses connect clusters



Switzerland

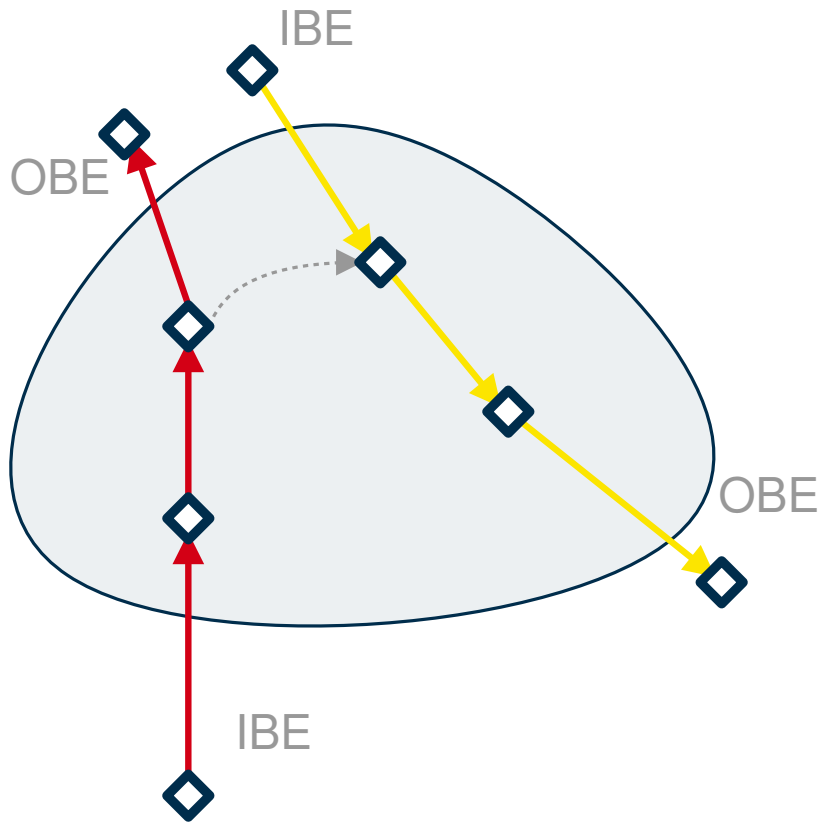
Customization

- compute *rank* $r(t) \in \mathbb{N}$ of every transfer
- rank denotes the highest level for “cell-crossing”



Customization

- compute *rank* $r(t) \in \mathbb{N}$ of every transfer
- rank denotes the highest level for “cell-crossing”
- compute all optimal journeys from *incoming border events* (IBEs) to OBEs

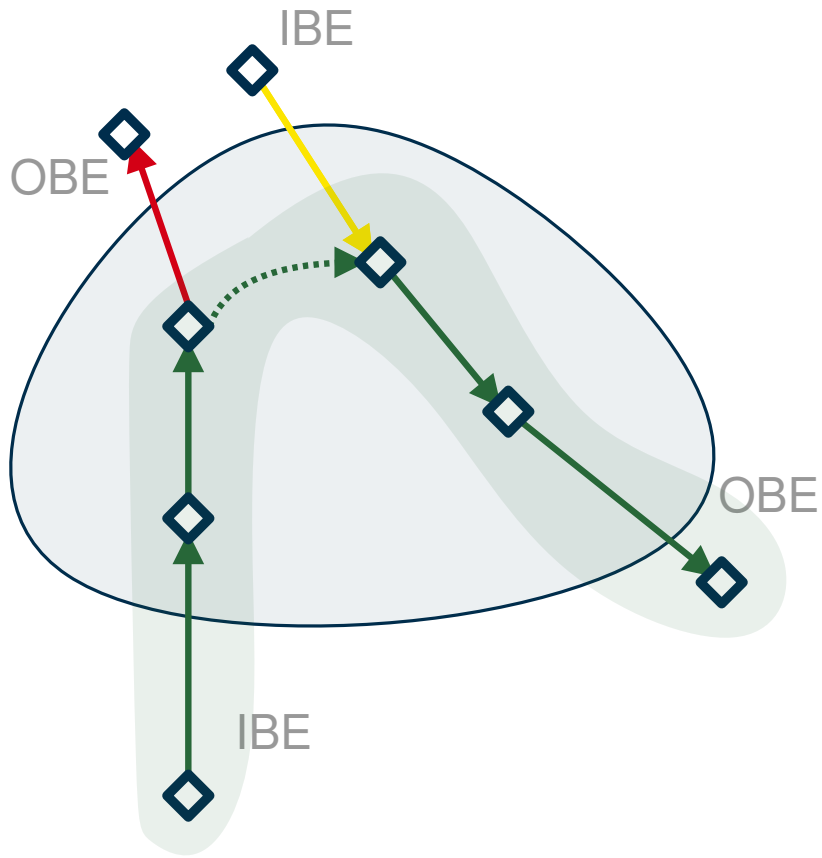


Customization

- compute *rank* $r(t) \in \mathbb{N}$ of every transfer
- rank denotes the highest level for “cell-crossing”

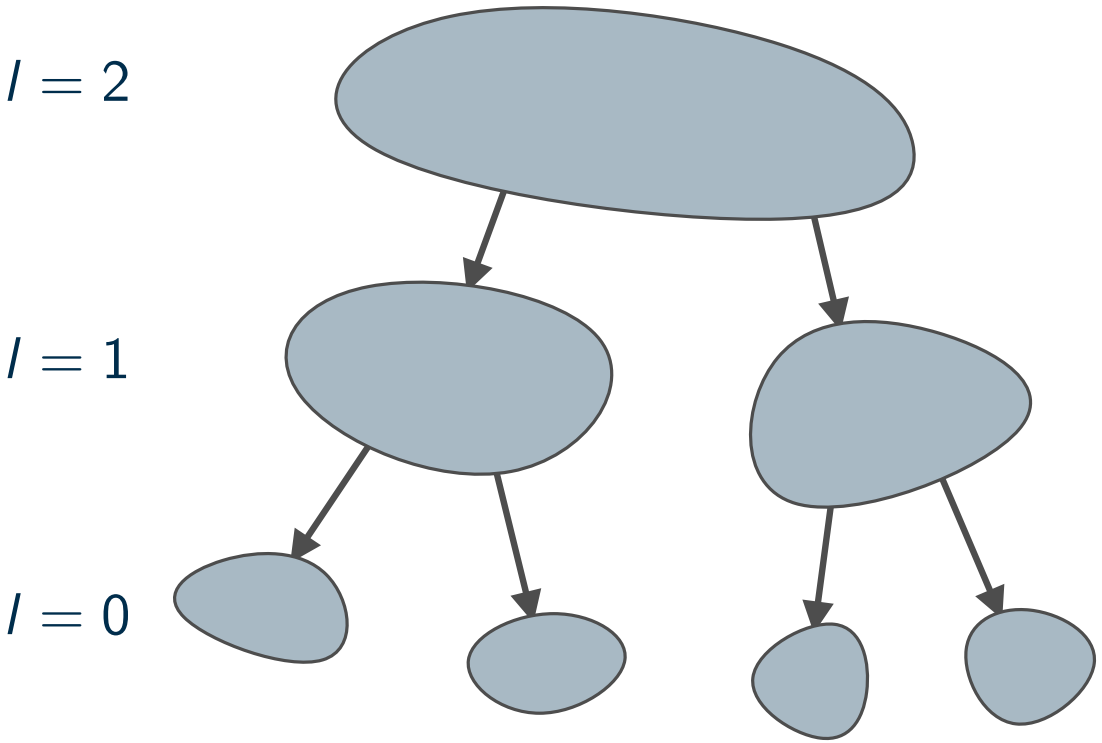
```
1: for all levels l ∈ [0, ℓ) do
2:     for all cells C on level l do
3:         for all IBEs e of C do
4:             run(e)
5:             for all “optimal” transfers t: r(t) ← l + 1
6:         end for
7:     end for
8: end for
```

- very efficient parallelization possible



Query

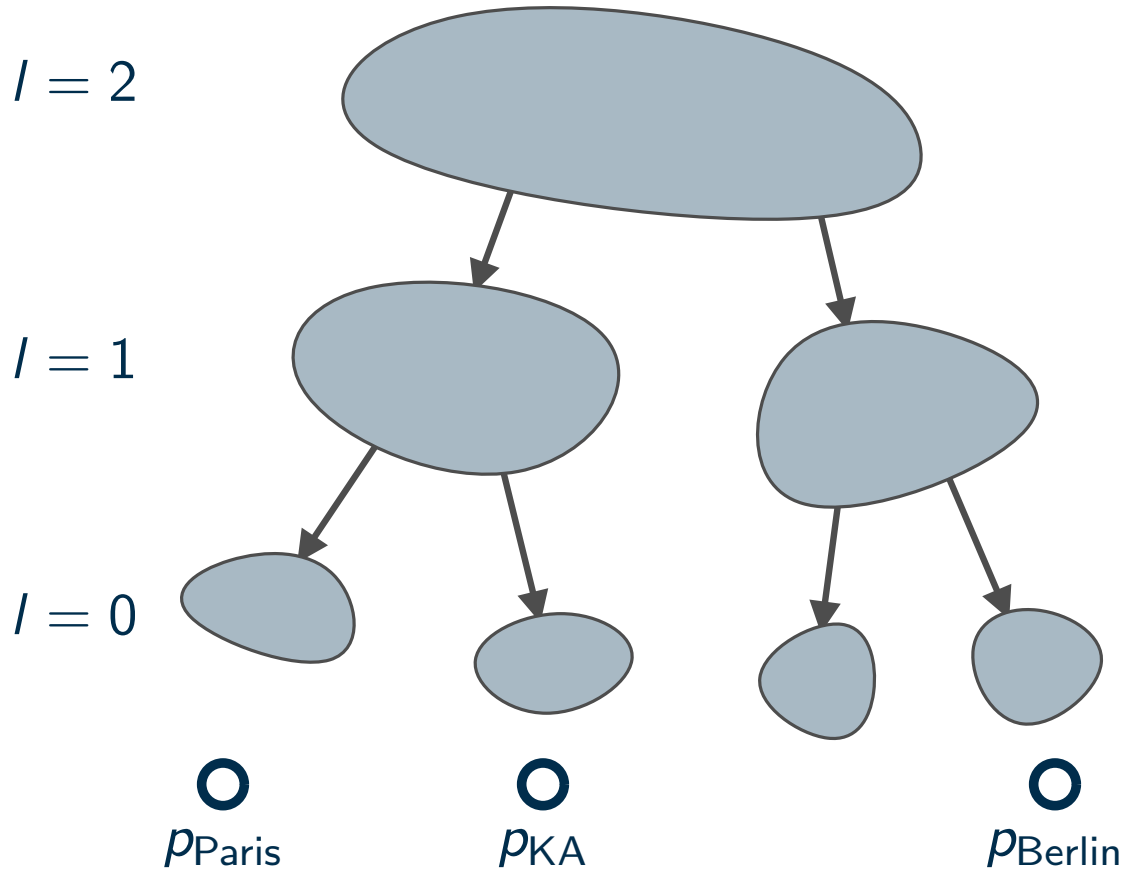
- main difference to TB: *prune* certain transfers
- pruning based on *lowest common level* LCL given two stops p_a, p_b



Query

- main difference to TB: *prune* certain transfers
- pruning based on *lowest common level* LCL given two stops p_a, p_b
- transfer t departing at Karlsruhe p_{KA}

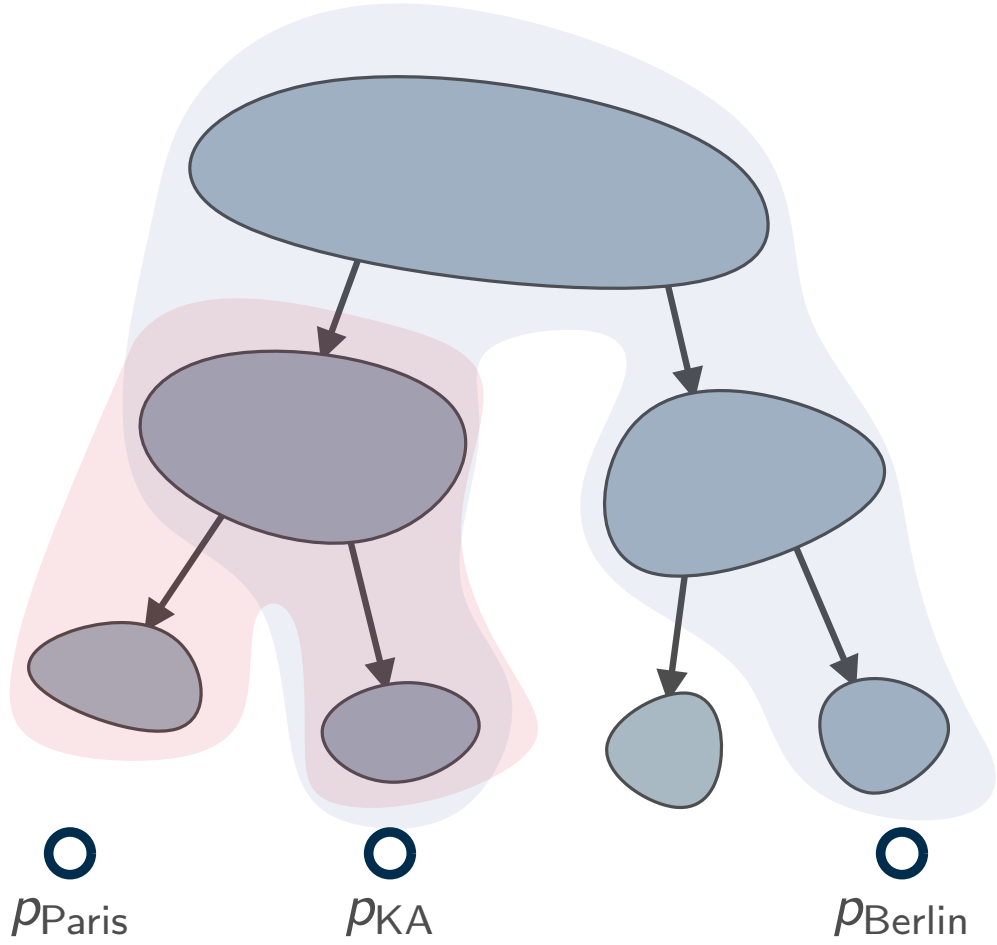
$$r(t) \geq \max\{\text{LCL}(p_{\text{Paris}}, p_{\text{KA}}), \text{LCL}(p_{\text{Berlin}}, p_{\text{KA}})\}$$



Query

- main difference to TB: *prune* certain transfers
- pruning based on *lowest common level* LCL given two stops p_a, p_b
- transfer t departing at Karlsruhe p_{KA}

$$
\begin{aligned}
r(t) &\geq \max\{\text{LCL}(p_{\text{Paris}}, p_{\text{KA}}), \text{LCL}(p_{\text{Berlin}}, p_{\text{KA}})\} \\
&\geq \max\{\text{LCL}(p_{\text{Paris}}, p_{\text{KA}}), \text{LCL}(p_{\text{Berlin}}, p_{\text{KA}})\} \\
&\geq \max\{1, 2\} \\
&\geq 2
\end{aligned}
$$



Experiments

- no standard benchmark available :(
- everyone runs their code on their machine and their datasets :(

Experiments

- no standard benchmark available :(
- everyone runs their code on their machine and their datasets :(

- either original code or reimplementations (except Scalable TP)
- used same machine, same datasets and same compiler
- modelled two consecutive weekdays

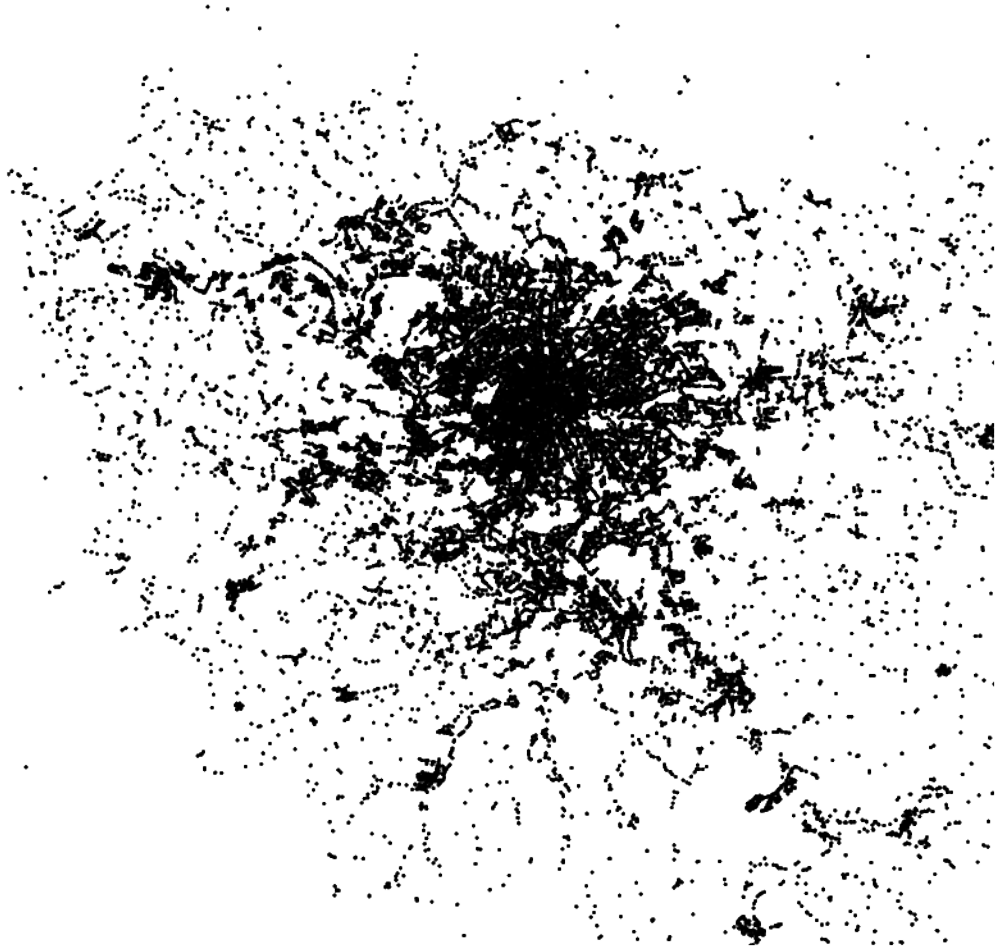
Datasets - Continent & Countries

	Europe	Germany	GB	Switzerland
Stops	1.346.013	435.550	260.150	29.045
Stop events	107.252.199	30.680.868	19.692.037	5.032.795
Lines	384.075	202.238	36.071	15.967
Trips	4.848.876	1.547.126	506.704	319.159
Footpaths	1.752.418	1.116.976	345.594	22.186
TB transfers	269.766.664	59.181.359	31.443.687	8.075.627
TB prepro.	8:55	1:23	0:24	0:04

Datasets - Metropolitain

	Paris	Berlin
Stops	41.757	25.843
Stop events	4.636.238	3.318.251
Lines	9.558	10.891
Trips	215.526	146.897
Footpaths	445.912	29.428
TB transfers	23.284.123	6.104.156
TB prepro.	0:17	0:07

Datasets



Graph Partitioner

- Mt-KaHyPar
  - very fast (seconds for Europe in 2^{16} cells)
  - best results (customization and query)

Graph Partitioner

- Mt-KaHyPar
  - very fast (seconds for Europe in 2^{16} cells)
  - best results (customization and query)
- Other partitioners
  - PUNCH / Buffoon
  - hierachical 2-d tree
  - hierachical 2-k means

very small balanced cuts on road networks

used by Scalable Transfer Patterns

Customization

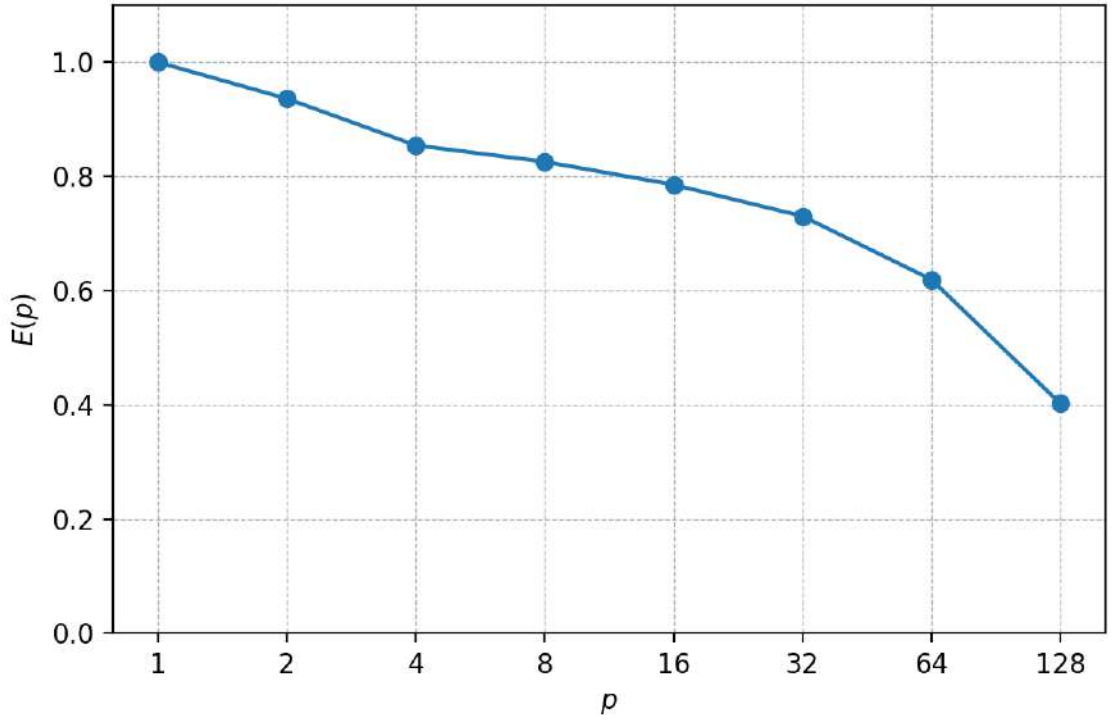
- evaluate 25%, 50%, 75% and 100% imbalance per level
  - choose the best query-performing settings

Customization

- evaluate 25%, 50%, 75% and 100% imbalance per level
  - choose the best query-performing settings

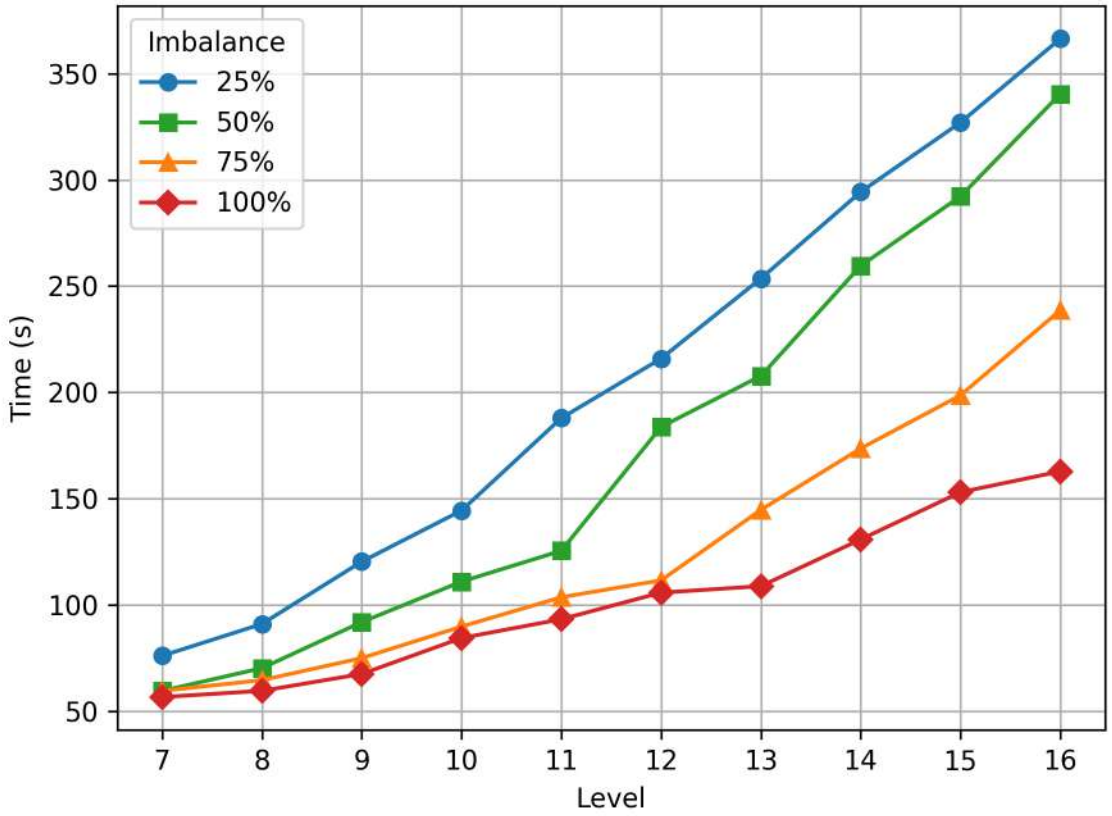
Germany 9 levels and 25% imbalance

- 1 thread: 11:55 mm:ss
- 2 thread: 06:22 mm:ss
- 4 thread: 03:29 mm:ss
- . . .
- 128 threads: 00:13 mm:ss



Customization

Europe 64 threads

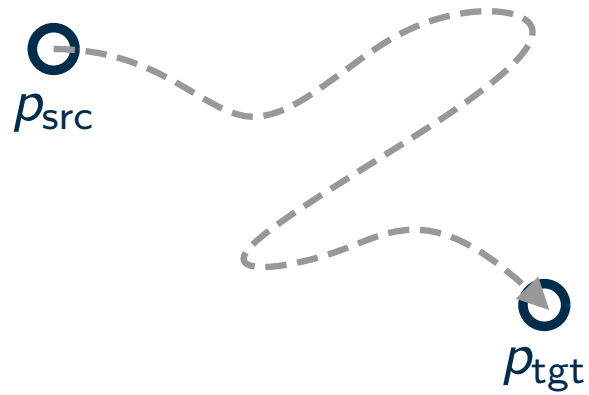


- more levels → more IBEs → takes more time
- more imbalance → fewer IBEs
  - more “difficult” cells

Query

Reported average query time over 10.000 queries:

- pick p_s, p_t uniformly at random
- pick departure time $\in$ [00:00, 24:00)



Comparison With State-Of-The-Art

Europe

Algorithm	Pareto	Query [μs]	Prepro. [hh:mm:ss]	k
TED	○	8.780.484	-	-
TDD	○	1.270.411	-	-
CSA	○	392.486	-	-
ACSA	○	38.510	00:02:16	4.096
RAPTOR	●	715.512	-	-
TB	●	222.834	00:05:26	-
T-REX	●	18.009	00:09:01	4.096

Comparison With State-Of-The-Art

Germany

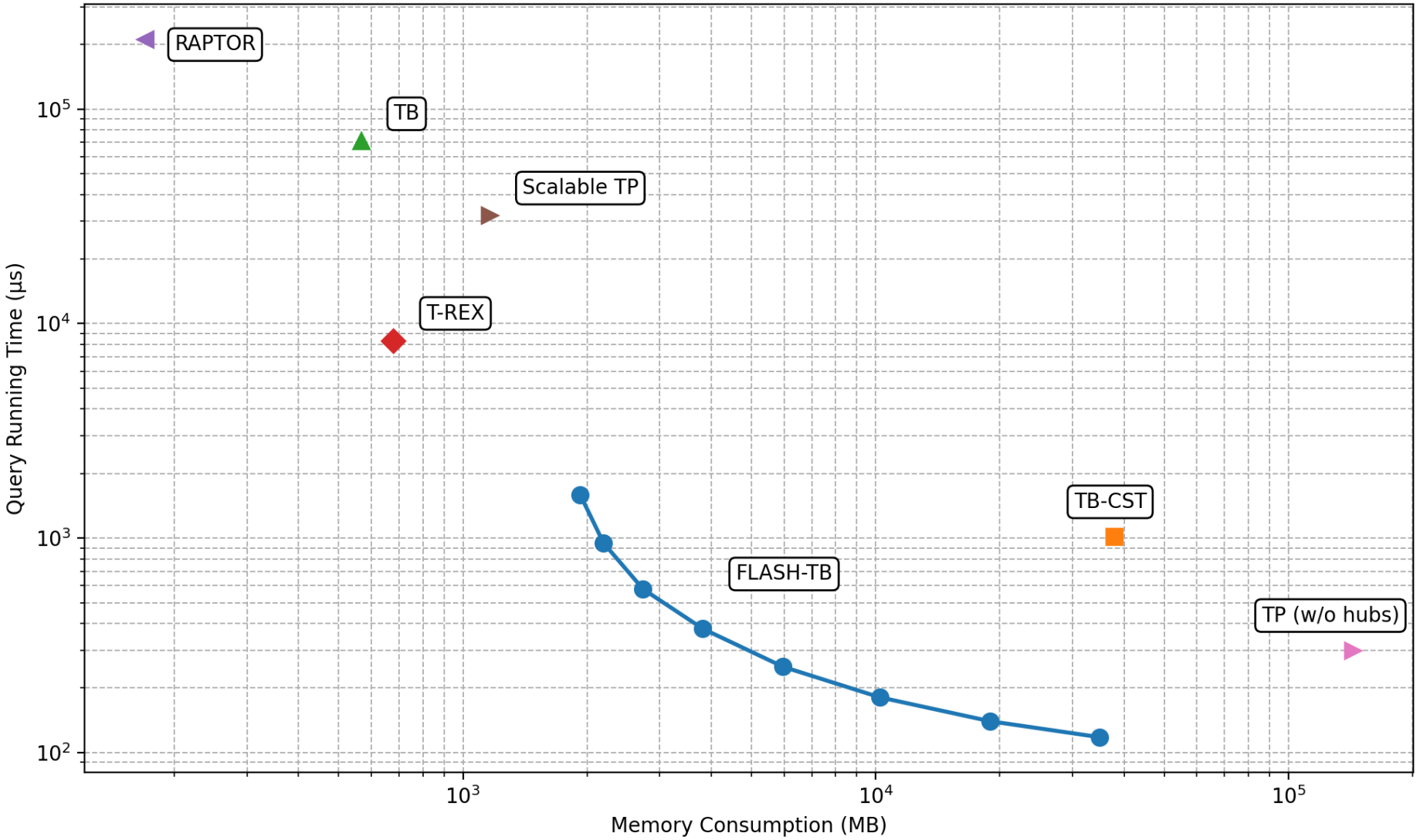
Algorithm	Pareto	Query [μs]	Prepro. [hh:mm:ss]	k
TED	○	1.466.681	-	-
TDD	○	499.849	-	-
CSA	○	83.101	-	-
ACSA	○	18.333	00:00:10	512
RAPTOR	●	212.255	-	-
TB	●	71.030	00:00:45	-
T-REX	●	8.331	00:00:58	512
TB-CST	●	1.017	09:27:42	-
FLASH-TB	●	127	32:47:49	8.192

Comparison With State-Of-The-Art

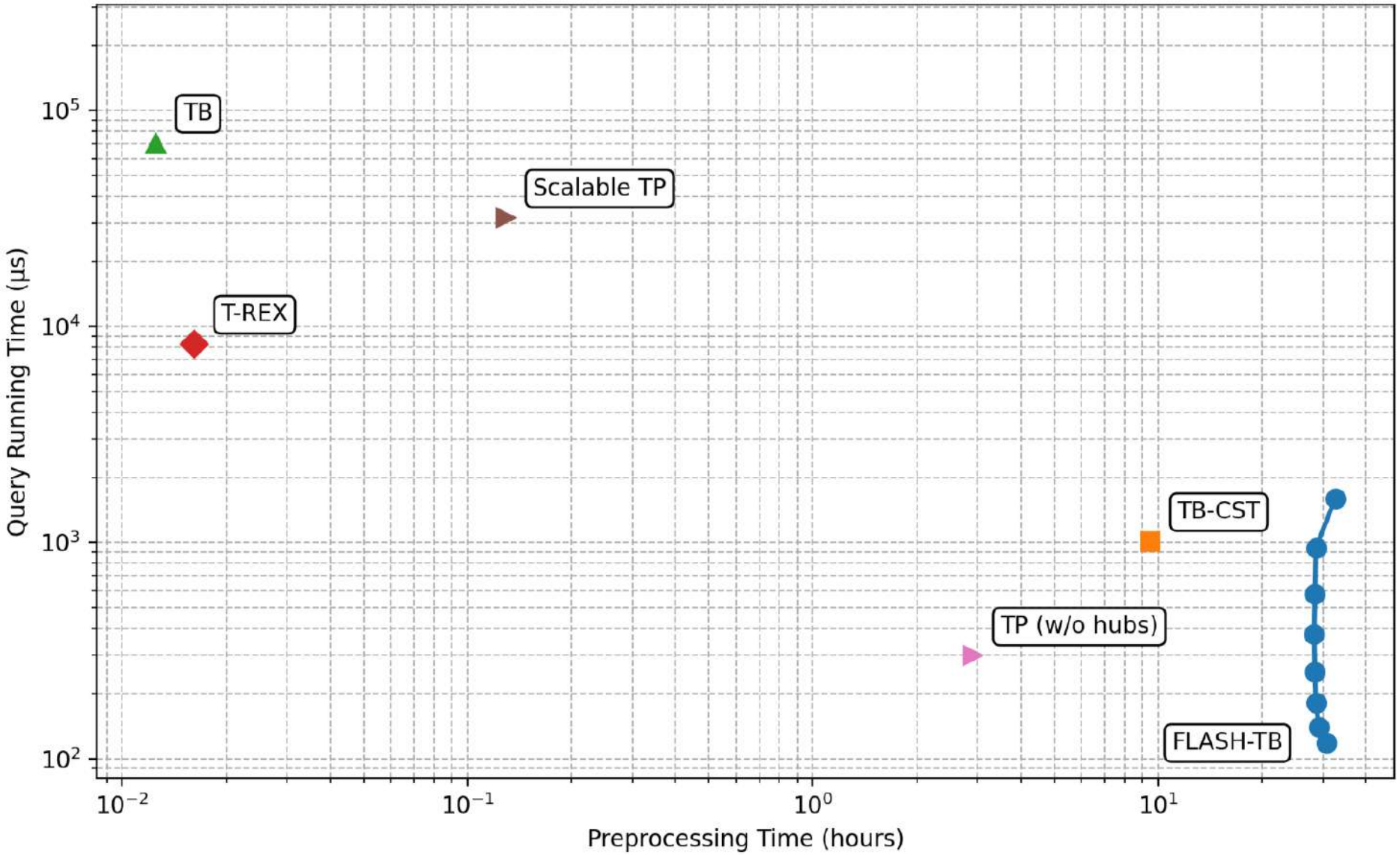
Paris

Algorithm	Pareto	Query [μs]	Prepro. [hh:mm:ss]	k
TED	○	249.700	-	-
TDD	○	17.985	-	-
CSA	○	3.280	-	-
ACSA	○	6.301	00:00:16	64
RAPTOR	●	8.562	-	-
TB	●	3.746	00:00:07	-
T-REX	●	2.018	00:00:18	1.024
TB-CST	●	36	00:04:00	-
FLASH-TB	●	25	00:19:42	16.384

Trade-Off - Germany



Trade-Off - Germany



Conclusion & Future Work

- efficient query algorithm, $< 20ms$ on Europe
- fast preprocessing
- low memory overhead

Conclusion & Future Work

- efficient query algorithm, $< 20ms$ on Europe
- fast preprocessing
- low memory overhead

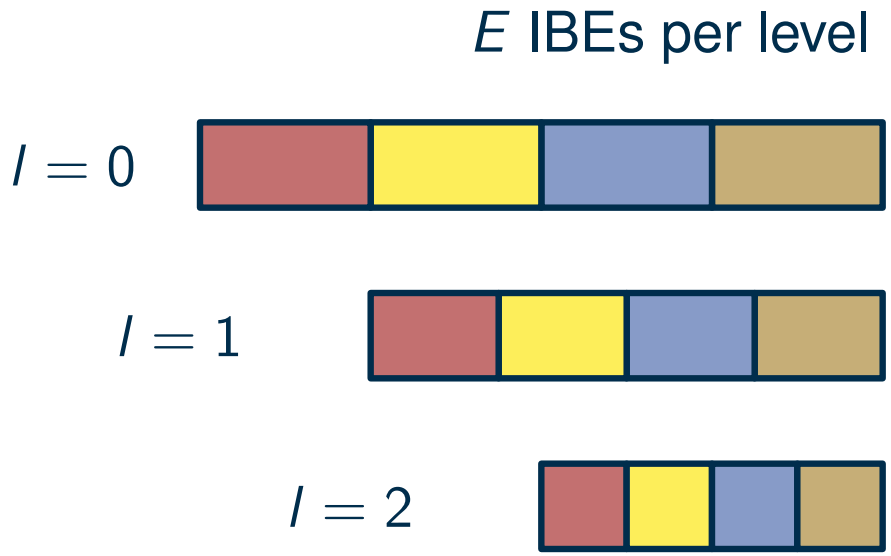
- evaluate delay updates
- add shortcuts spanning over cells
- processing “longer” timetables

Thank you!

Parallelized Customization

- parallelization within every level
- all IBEs on current level are collected, and distributed among threads

```
1: for all levels l ∈ [0, ℓ) do
2:     E ← all IBEs on level l
3:     for all e ∈ E do in parallel
4:         run(e)
5:         for all "optimal" transfers t: r(t) ← l + 1
6:     end for
7: end for
```



Fixed-Departure Time Query

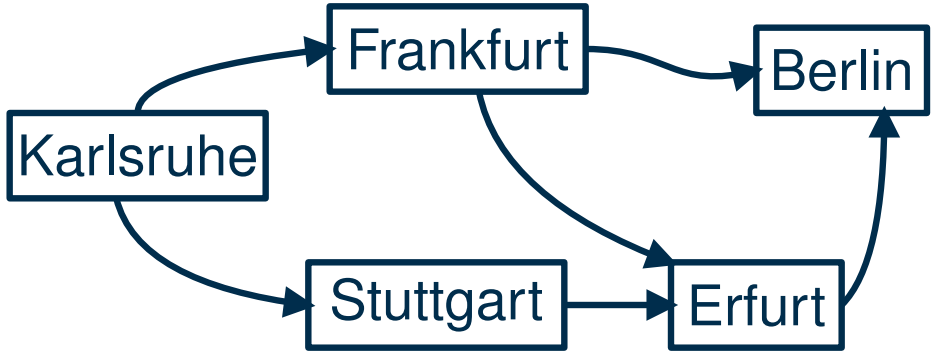
Network	Algorithm	Query [μs]	Prepro. [mm:ss]	k	Imbalance [%]
Europe	TB	222.834	05:26	-	-
	T-REX	18.009	09:01	4.096	25
Germany	TB	71.030	00:45	-	-
	T-REX	8.331	00:58	512	25
Great Britain	TB	17.477	00:24	-	-
	T-REX	2.293	00:30	2.048	50
Switzerland	TB	4.605	00:04	-	-
	T-REX	924	00:06	1.024	25

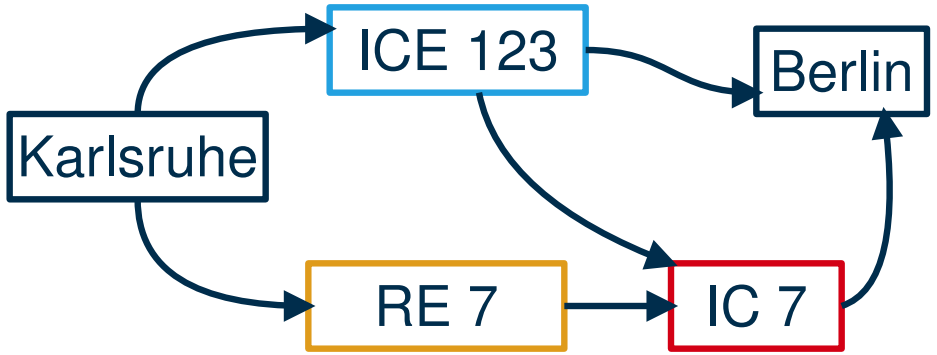
Fixed-Departure Time Query

Network	Algorithm	Query [μs]	Prepro. [mm:ss]	k	Imbalance [%]
Paris	TB	3.746	00:07	-	-
	T-REX	2.018	00:18	1.024	25
Berlin	TB	3.055	00:04	-	-
	T-REX	1.046	00:05	512	50

TP, FLASH-TB & TB-CST

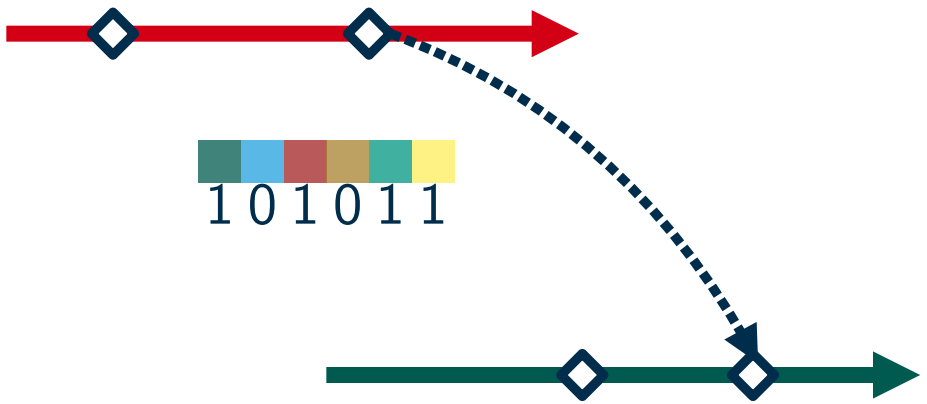
- heavy preprocessing and memory overhead
- Transfer Patterns and TB-CST
  - abstract (time independent) representation of optimal journeys
  - query very fast due to small search space





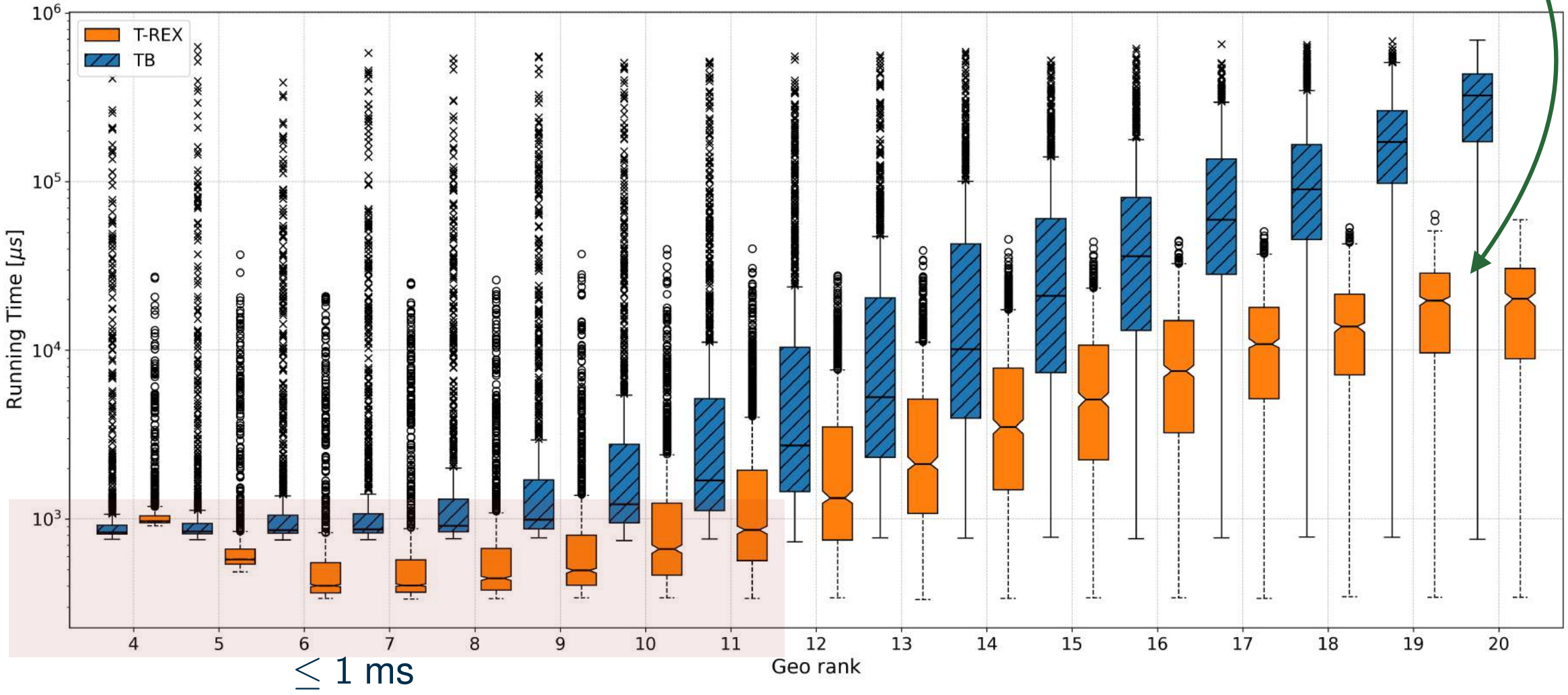
TP, FLASH-TB & TB-CST

- heavy preprocessing and memory overhead
- Transfer Patterns and TB-CST
  - abstract (time independent) representation of optimal journeys
  - query very fast due to small search space
- FLASH-TB
  - no additional datastructure or graph
  - (Goaldirected) Arc-Flags on transfer



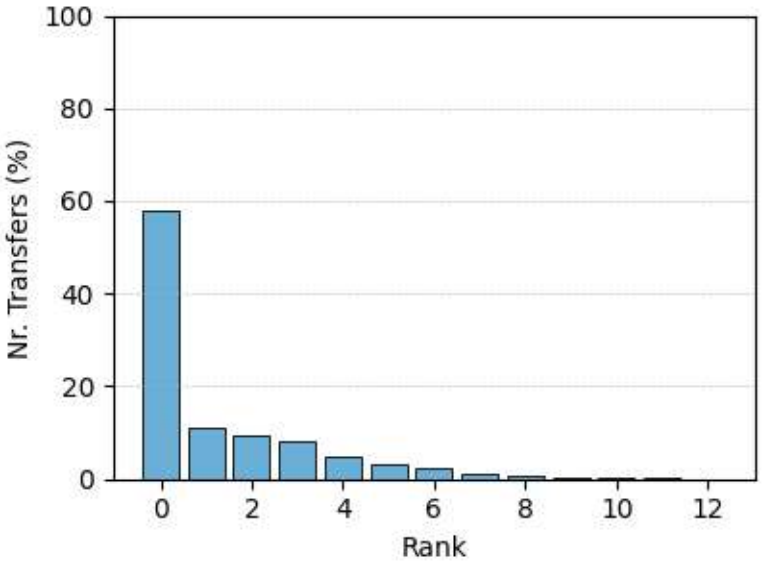
Geo-Rank Query - Europe

avg fixed-departure time query 18 ms

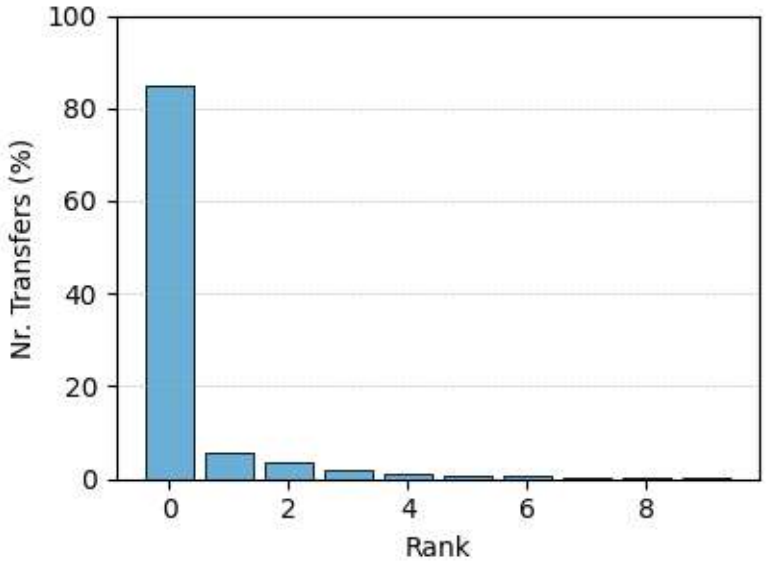


$\leq$ 1 ms

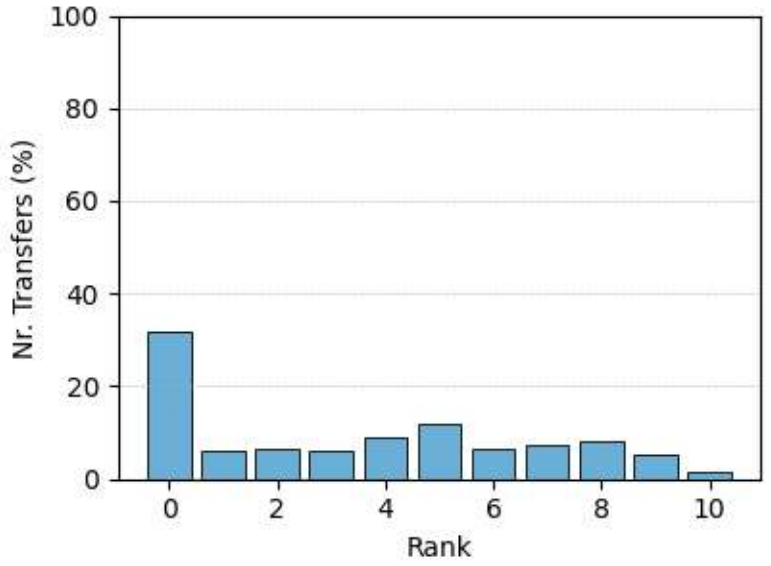
Rank Histogram



Europe



Germany



Paris

Efficient LCL Check

```
void enqueue(const Edge edge, const size_t parent) {
    const EdgeLabel &label = edgeLabels[edge];

    if (reachedIndex.alreadyReached(label.trip, label.stopEvent))
        return;

   reachedIndex.update(label.trip, StopIndex(label.stopEvent));

    if (((label.cellId ^ sourceCellId) >> label.localLevel) &&
        ((label.cellId ^ targetCellId) >> label.localLevel)) return;

    queue[queueSize] = TripLabel(
        StopEventId(label.stopEvent + label.firstEvent),
        StopEventId(label.firstEvent + reachedIndex(label.trip)),
            parent);
    ++queueSize;
  }
```