OST
Eastern Switzerland
University of Applied Sciences

Efficient Public Transit Routing with RAPTOR

A Production-Ready Solution Leveraging GTFS Data

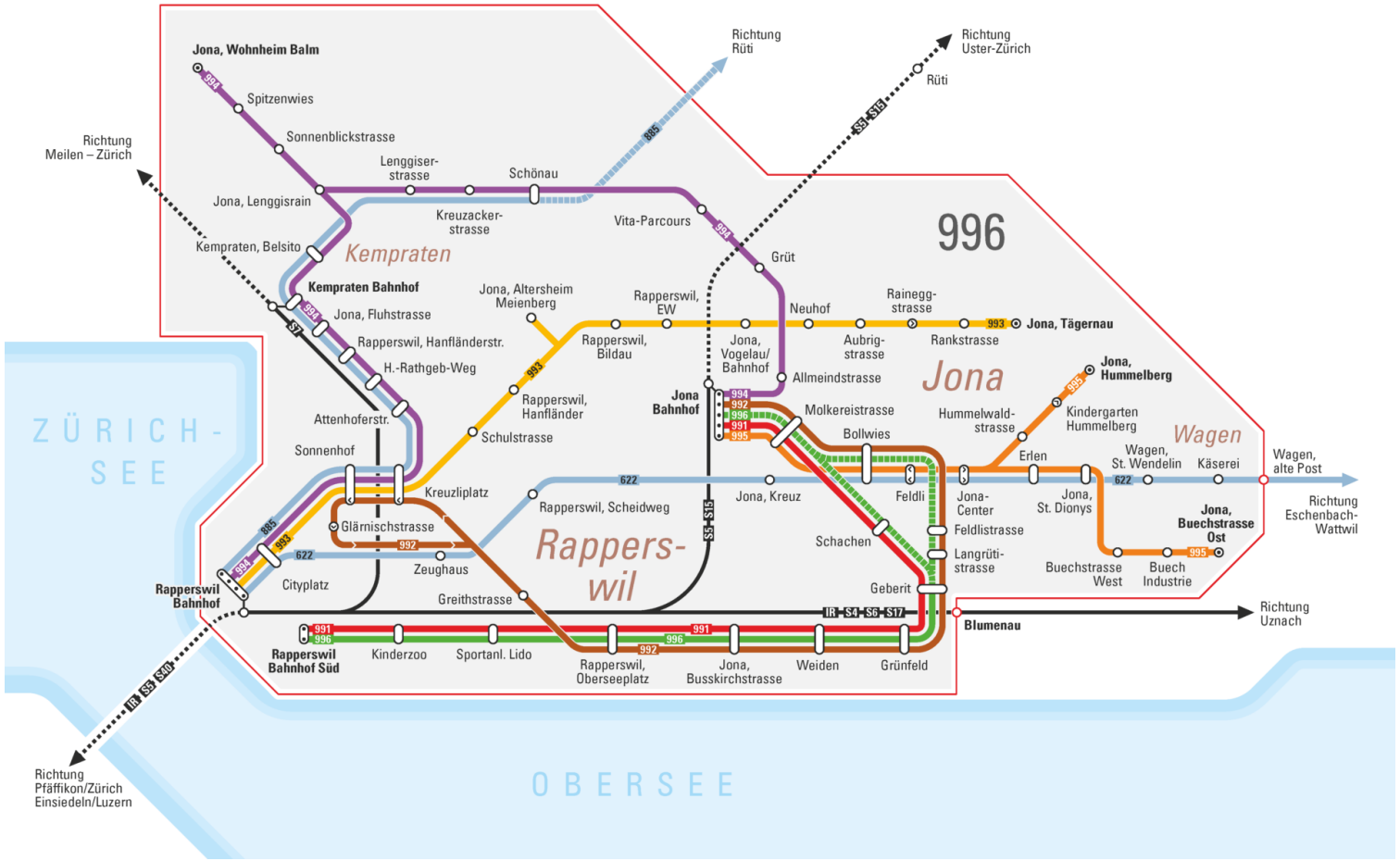
Michael Brunner, Lukas Connolly, Merlin Unterfinger
Supervisor: Thomas Letsch

24 October 2024

MAS Software Engineering 2022 - 2024

Agenda

- Project Management
- Analysis
- Design
- Implementation
- Demo
- Results
- Conclusion



Source: https://www.vzo.ch/stadtbus-rapperswil-jona

Team

Lukas Connolly
Senior Research Engineer

Zimmer Biomet
Computational Biomechanics
Sulzerallee 8
CH-8404 Winterthur

lukas.connolly@zimmerbiomet.com

Michael Brunner
Software Engineer

Cadwork Holz AG
Informatik
Industriestrasse 28
CH-9100 Herisau

brunner@cadwork.swiss

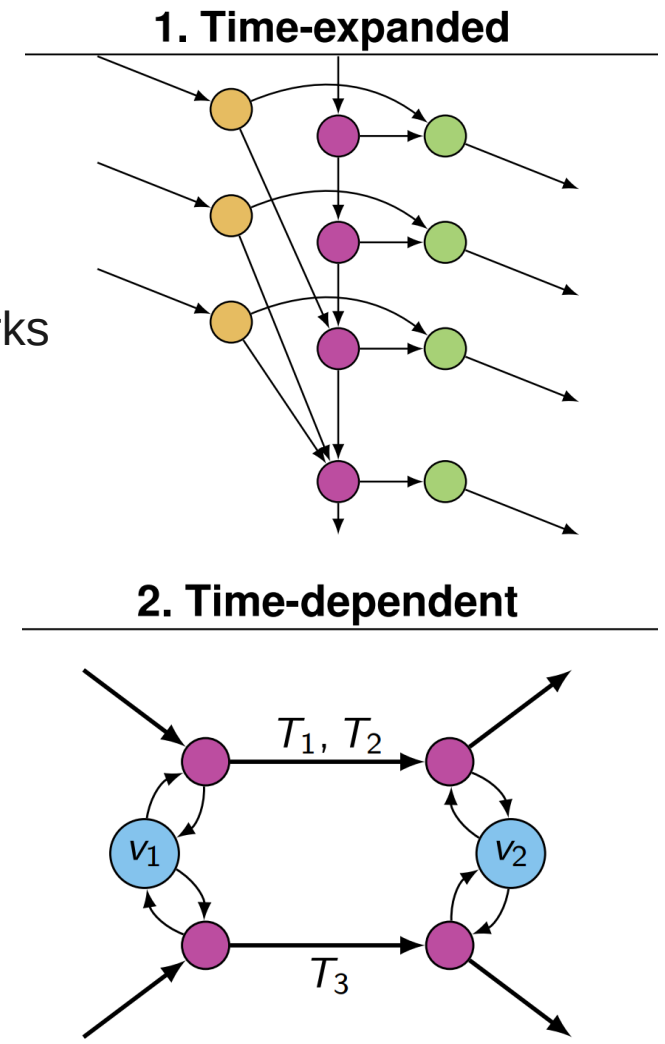
Merlin Unterfinger
Subject Matter Expert

SBB AG
Corporate Development
Hilfikerstrasse 1
CH-3000 Bern 65

merlin.unterfinger@sbb.ch

Motivation

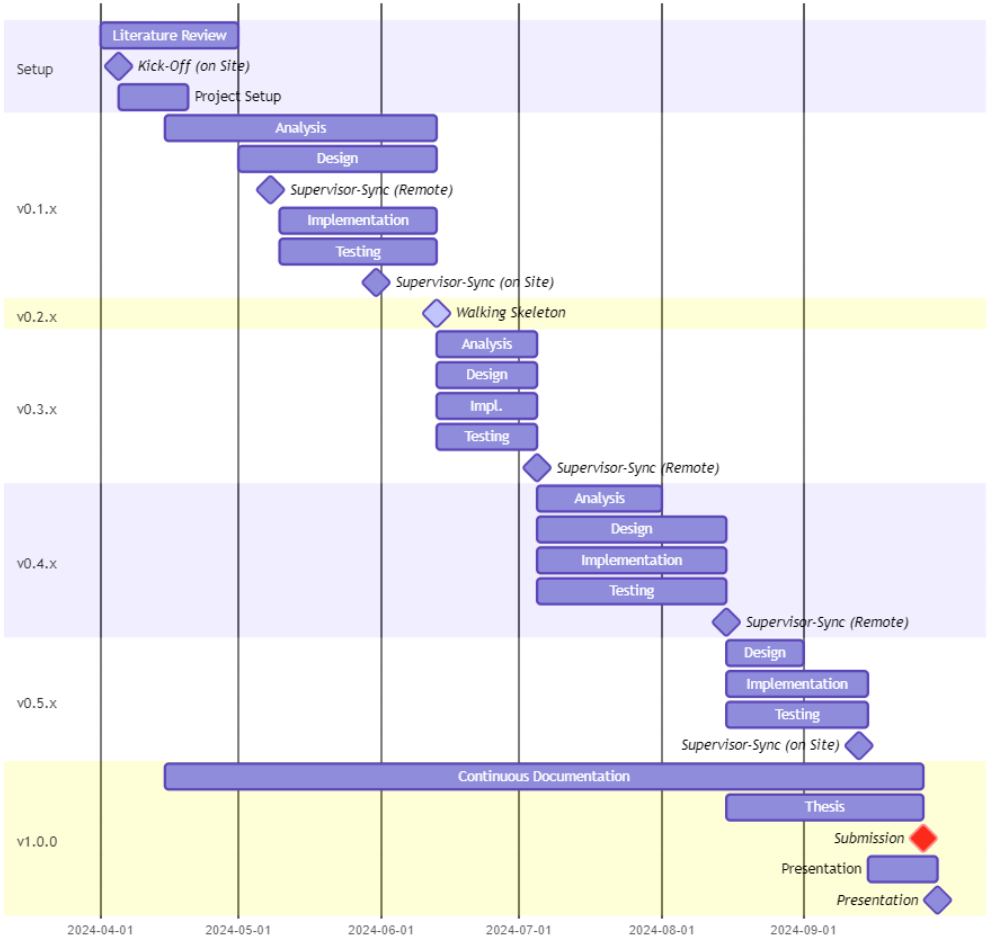
- Public transit routing is a **challenging topic**
  - Time-dependent transit schedules add complexity
  - Traditional graph-based algorithms struggle with large-scale networks
  - The RAPTOR algorithm offers efficient, low-preprocessing routing
- **Open Data** availability
  - General Transit Feed Specification (GTFS) standard
  - Many national transit agencies publish timetables
- Project integrates **software engineering** skills
  - Algorithms and data structures, architecture
  - Language-specific optimizations (Java, C++)



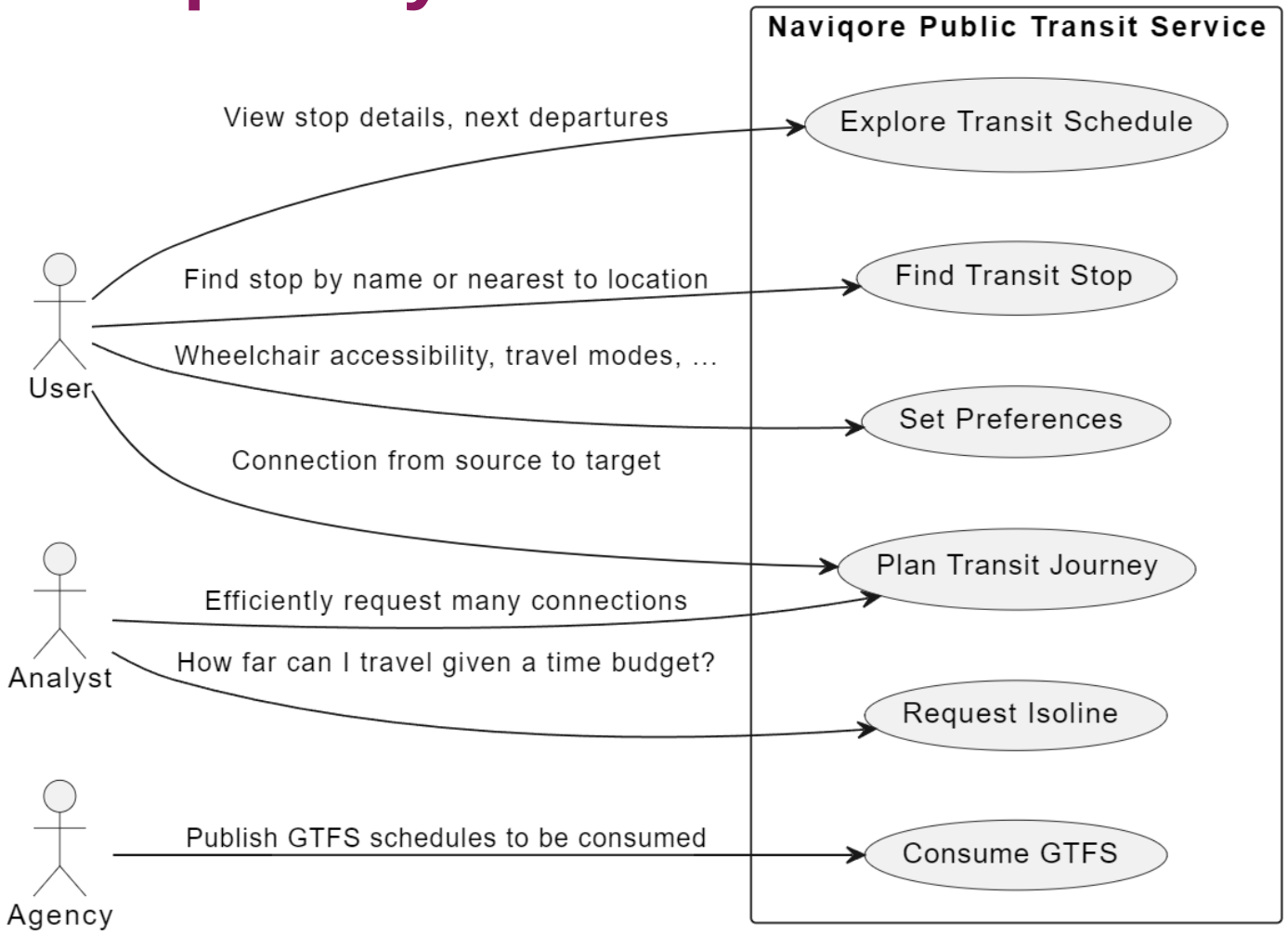
Source: Jonas Sauer (2021): Public Transit Journey Planning

Software Development Process

- Project structured in **3 phases**
  - Walking skeleton (*v0.2.0*)
  - Main implementation (*v1.0.0*)
  - Feature freeze
- **Scrum-like** development setup
  - Sprints of 1 month
  - Weeklies, planning, demo & review
  - Task and time management in Jira
- Written **guidelines**
  - Code style, Git & release process
  - Comprehensive pull request reviews



Use Cases – Naviqore System



Requirements – Naviqore System

- Structure: `{TYPE}-{COMPONENT}-{PRIORITY}{NUMBER}`

Requirement Types
UC: Use case
NF: Non-function requirement

Components
SE: Service
SC: Schedule
RO: Routing
VW: Viewer

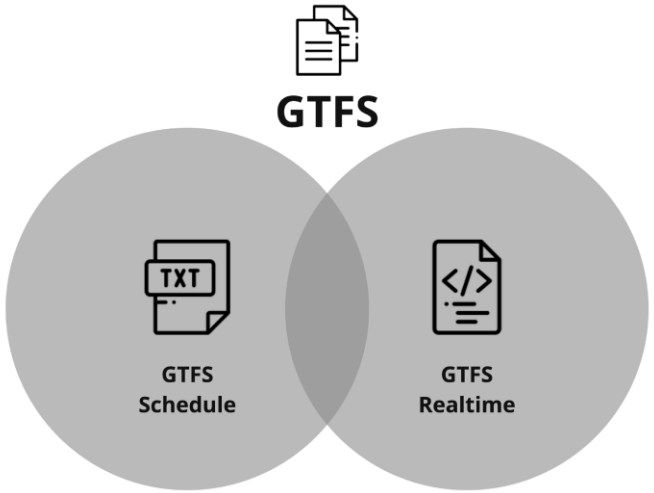
Priority Levels
M: Must-have
S: Should-have
N: Nice-to-have

- All must-have, and most of should-have requirements were satisfied

	Service	Schedule	Routing	Viewer	*Total*
Must-have	7 (7)	4 (4)	5 (5)	1 (1)	*17 (17)*
Should-have	4 (4)	1 (0)	7 (6)	1 (1)	*13 (11)*
Nice-to-have	4 (1)	1 (0)	4 (1)	7 (2)	*16 (4)*
Total	*15 (12)*	*6 (4)*	*16 (12)*	*9 (4)*	*46 (32)*

Analysis

GTFS (General Transit Feed Specification)



- Text files
- **Fixed** schedules, stops, and routes
- Timetables, stop locations, fare information

- Protocol buffer
- **Live** updates on transit services
- Delays, vehicle locations, service alerts.

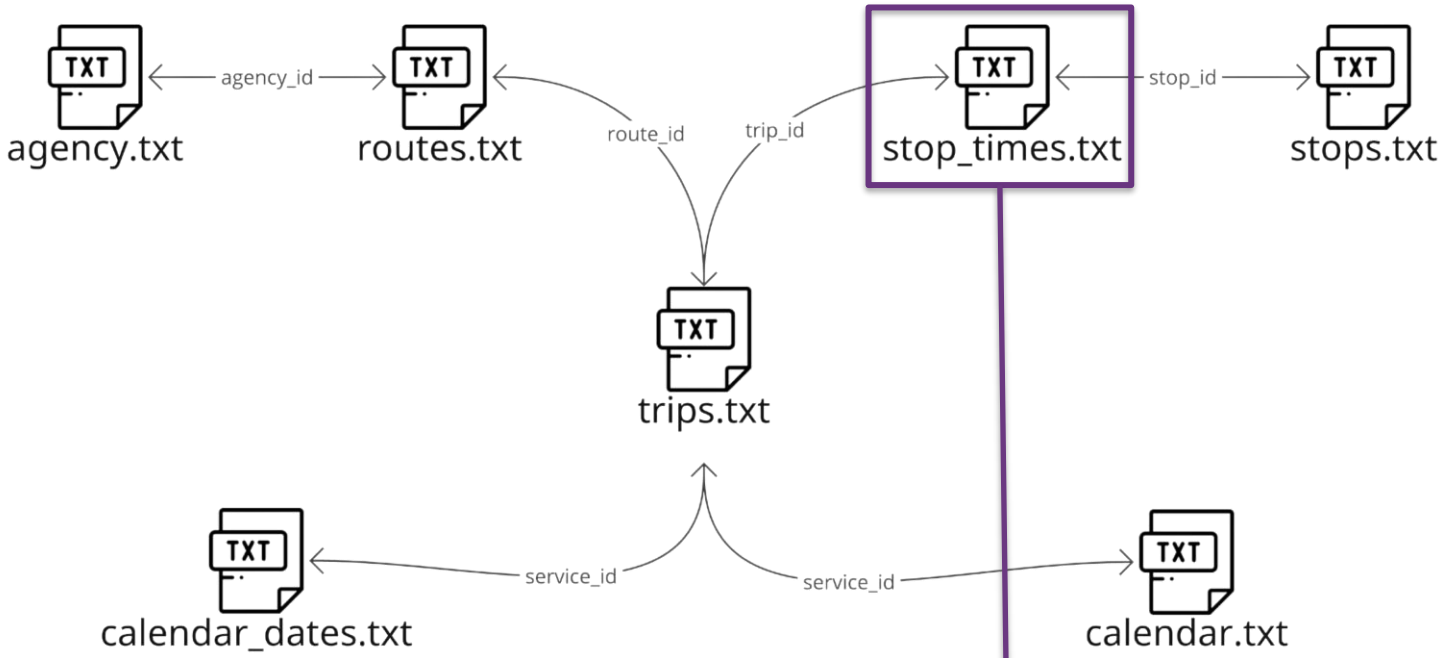
Source: https://opentransportdata.swiss/en/cookbook/gtfs/

GTFS – Static

Timetable of Switzerland:

File	Size
agency.txt	43.0 KB
calendar.txt	3.3 MB
calendar_dates.txt	194.0 MB
routes.txt	208.0 KB
stop_times.txt	1.4 GB
stops.txt	6.4 MB
transfers.txt	1.5 MB
trips.txt	136.0 MB
Total	***1.7 GB***

Source: https://opentransportdata.swiss/en/group/timetables-gtfs




```
trip_id, arrival_time, departure_time, stop_id, stop_sequence, pickup_type, drop_off_type
"1.TA.91-10-A-j24-1.1.H", "11:37:00", "11:37:00", "8503054:0:1", "1", "0", "0"
"1.TA.91-10-A-j24-1.1.H", "11:38:00", "11:38:00", "8503053:0:1", "2", "0", "0"
"1.TA.91-10-A-j24-1.1.H", "11:39:00", "11:39:00", "8503052:0:1", "3", "0", "0"
"1.TA.91-10-A-j24-1.1.H", "11:41:00", "11:42:00", "8503051:0:1", "4", "0", "0"
"1.TA.91-10-A-j24-1.1.H", "11:44:00", "11:44:00", "8503090:0:1", "5", "0", "0"
"1.TA.91-10-A-j24-1.1.H", "11:47:00", "11:47:00", "8503088:0:2", "6", "0", "0"
"1.TA.91-10-B-j24-1.2.H", "07:43:00", "07:43:00", "8505307:0:1", "1", "0", "0"
```

Source: https://gtfs.org/getting_started/create/

RAPTOR Data Structure

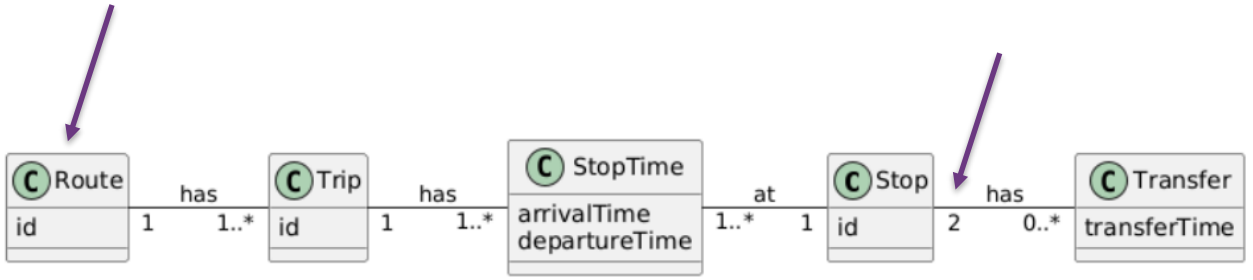


Figure 3.4: Domain model for RAPTOR.

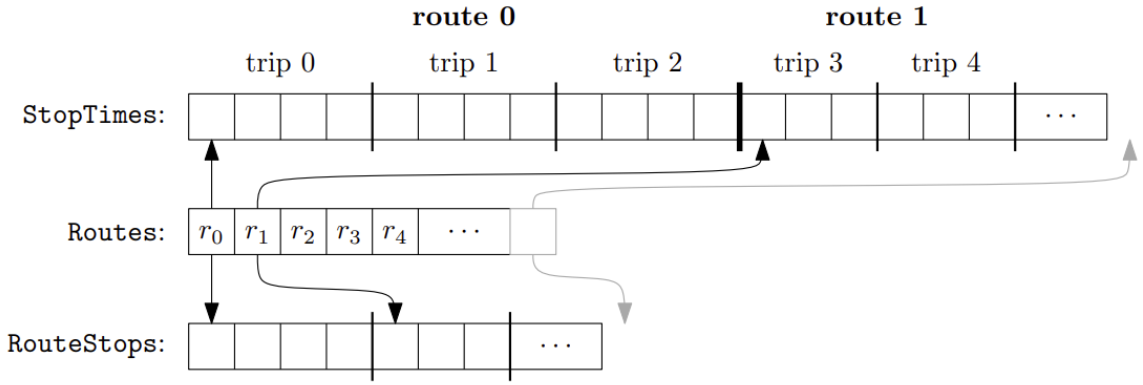


Figure 3: Illustration of the adjacency structure of routes.

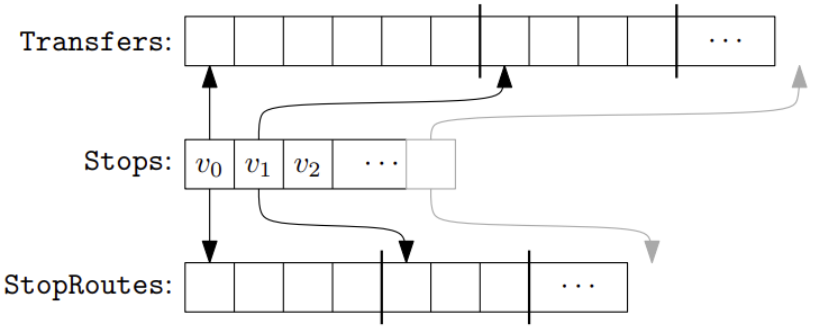
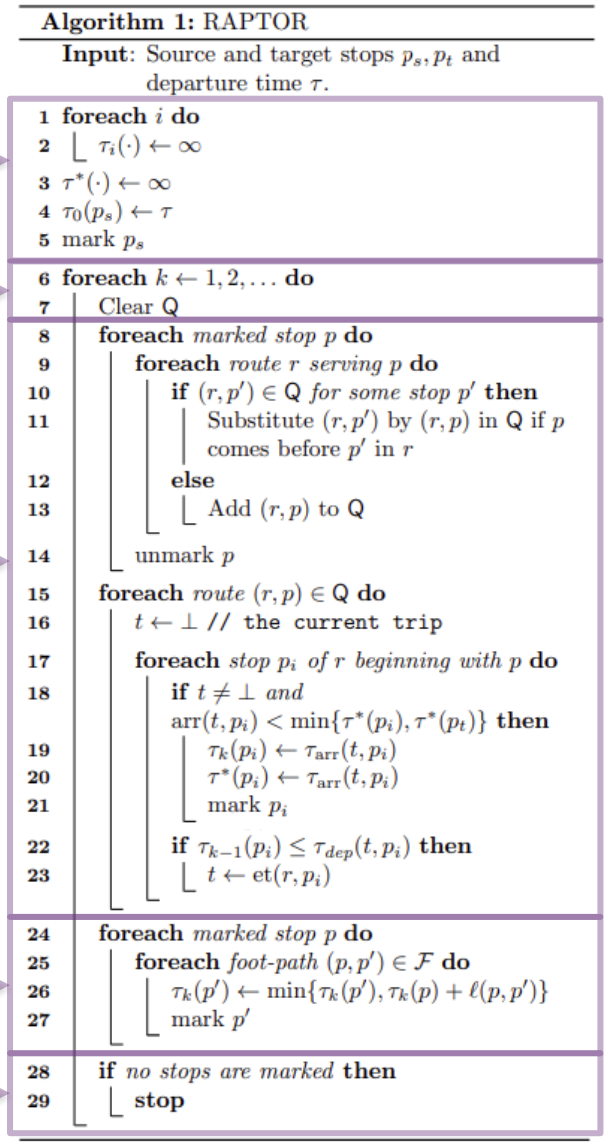


Figure 4: Illustration of the adjacency structure of stops.

Source: https://doi.org/10.1137/1.9781611972924.13

RAPTOR Algorithm

- **Initialization**: Mark first stop
- Initial **footpath relaxation**
- Main Loop: **Round-based search**
  - Add new round
  - **Scan** routes, mark improved stops
  - **Relax** footpaths, mark improved stops
- **Termination**: End when no improvements possible
- **Reconstruct**: Best route



Algorithm 1: RAPTOR

Input: Source and target stops p_s, p_t and departure time τ.

```
1  foreach i do
2      τ_i(·) ← ∞
3  τ*(·) ← ∞
4  τ_0(p_s) ← τ
5  mark p_s
6  foreach k ← 1, 2, ... do
7      Clear Q
8      foreach marked stop p do
9          foreach route r serving p do
10             if (r, p') ∈ Q for some stop p' then
11                 Substitute (r, p') by (r, p) in Q if p
                   comes before p' in r
12             else
13                 Add (r, p) to Q
14         unmark p
15     foreach route (r, p) ∈ Q do
16         t ← ⊥ // the current trip
17         foreach stop p_i of r beginning with p do
18             if t ≠ ⊥ and
               arr(t, p_i) < min{τ*(p_i), τ*(p_t)} then
19                 τ_k(p_i) ← τ_arr(t, p_i)
20                 τ*(p_i) ← τ_arr(t, p_i)
21                 mark p_i
22             if τ_{k-1}(p_i) ≤ τ_dep(t, p_i) then
23                 t ← et(r, p_i)
24     foreach marked stop p do
25         foreach foot-path (p, p') ∈ F do
26             τ_k(p') ← min{τ_k(p'), τ_k(p) + ℓ(p, p')}
27             mark p'
28     if no stops are marked then
29         stop
```

Accumulate routes serving marked stops from previous round

Traverse each route

Can the label be improved in this round? Includes local and target pruning

Relax foot-paths on marked stops

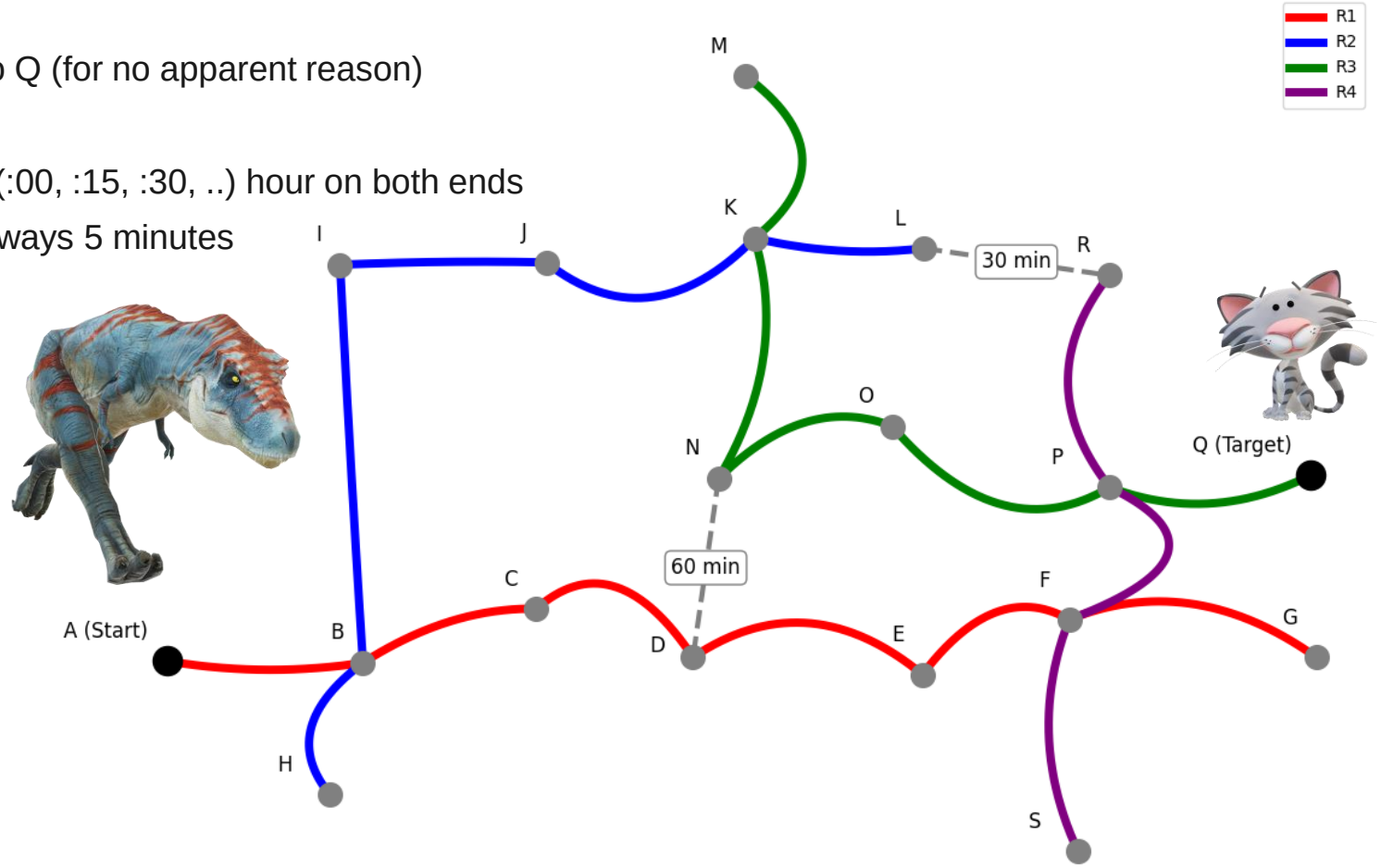
Stopping criterion

Source: https://doi.org/10.1137/1.9781611972924.13

OST

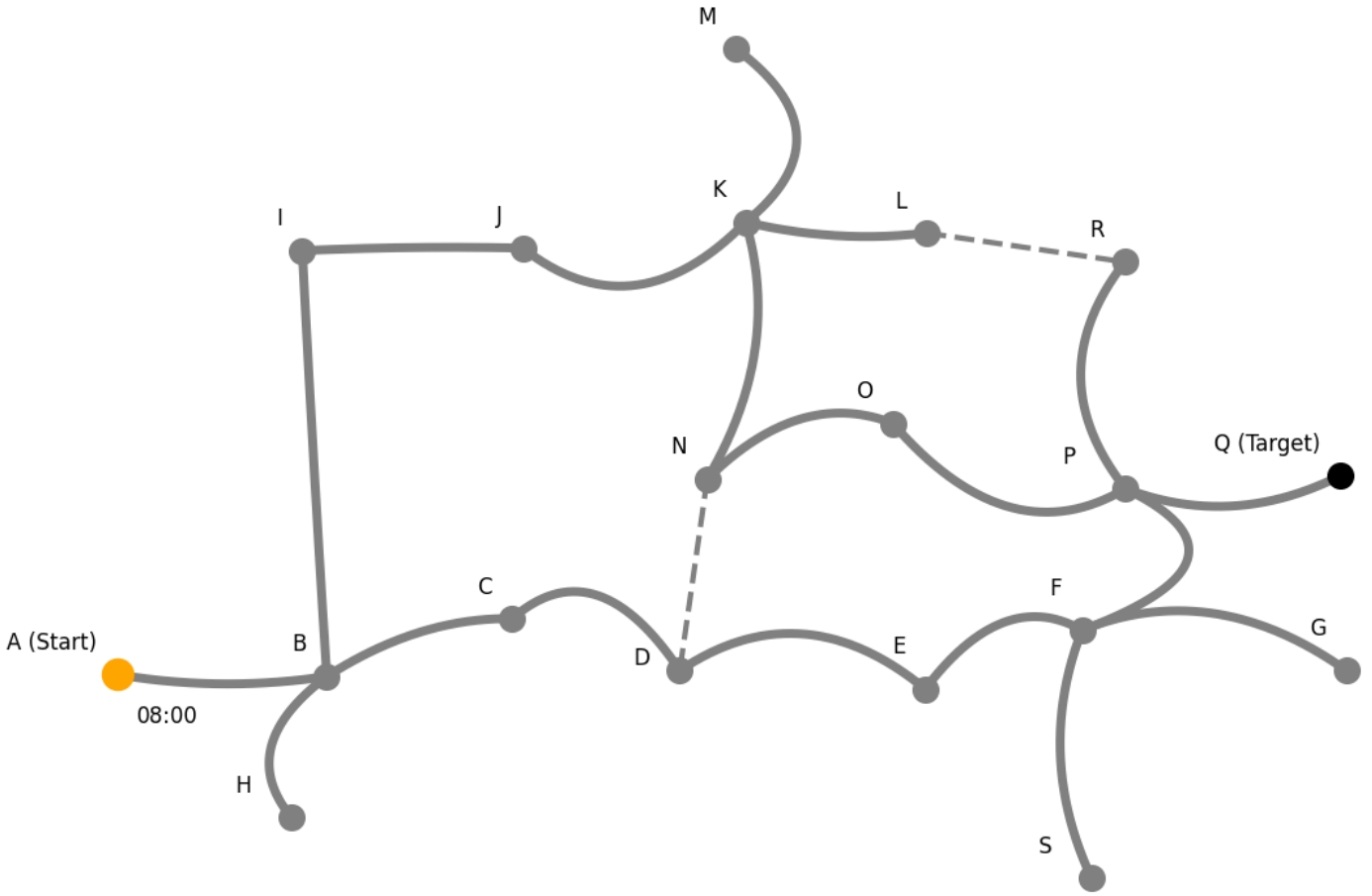
RAPTOR Algorithm

- The Raptor wants to get from A to Q (for no apparent reason)
- The current time is 08:00 AM
- All routes start every 15 minutes (:00, :15, :30, ..) hour on both ends
- The time between each stop is always 5 minutes



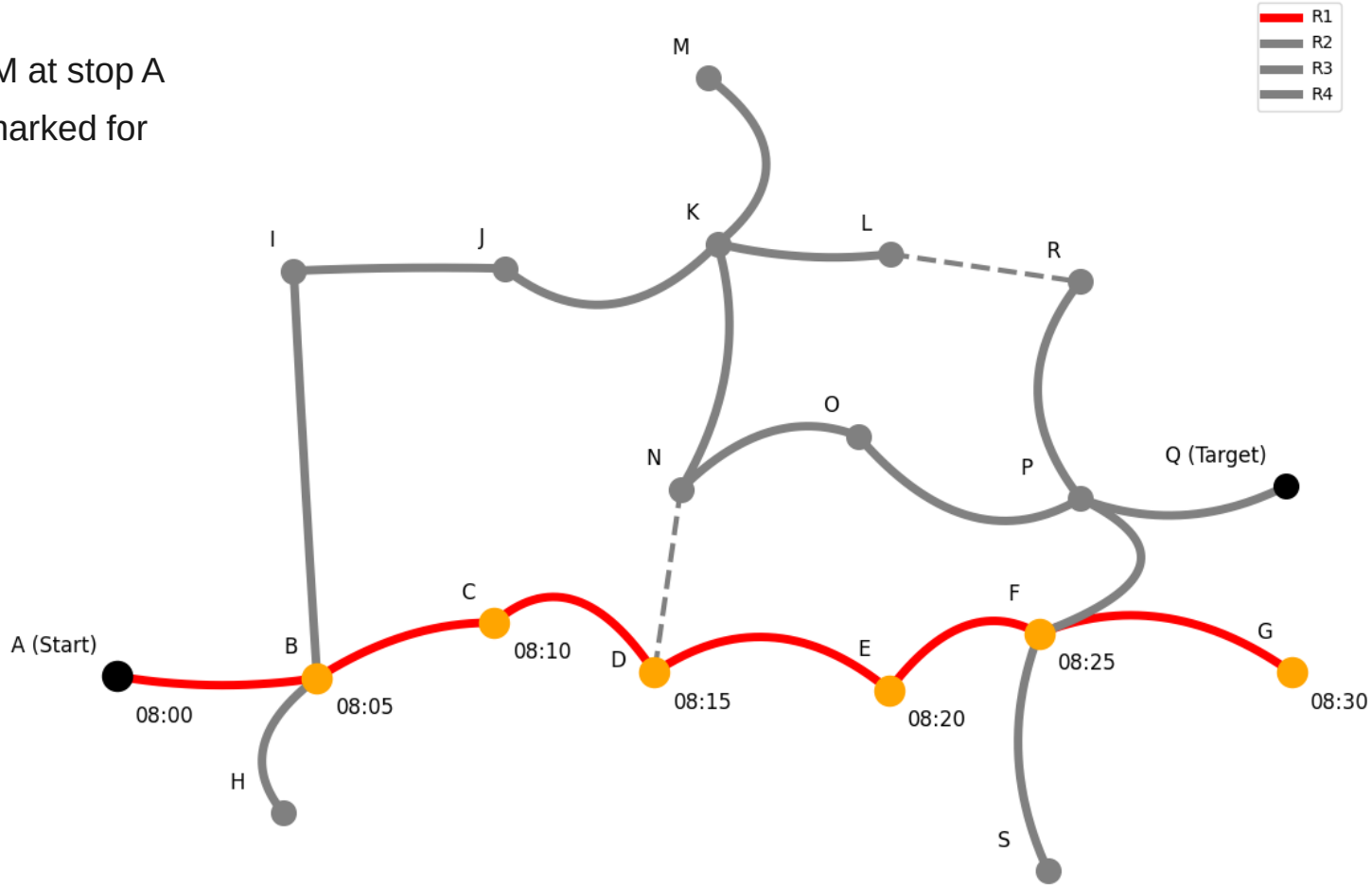
RAPTOR Algorithm (Round 0)

- Stop A is marked
- No transfers are possible from A



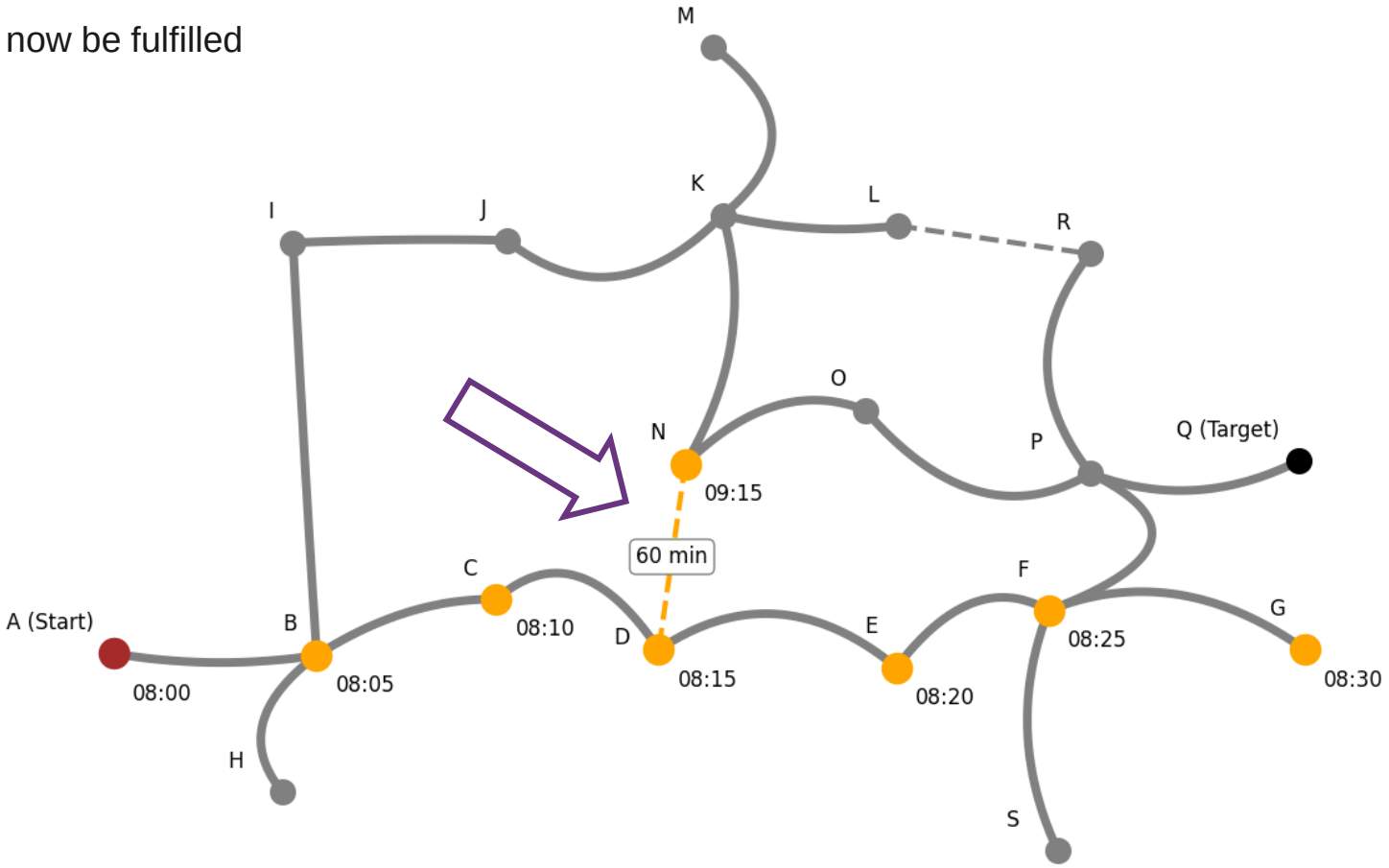
RAPTOR Algorithm (Round 1 – Route Scanning)

- Route R1 can be entered at 08:00 AM at stop A
- All stops along the line can now be marked for next round scanning



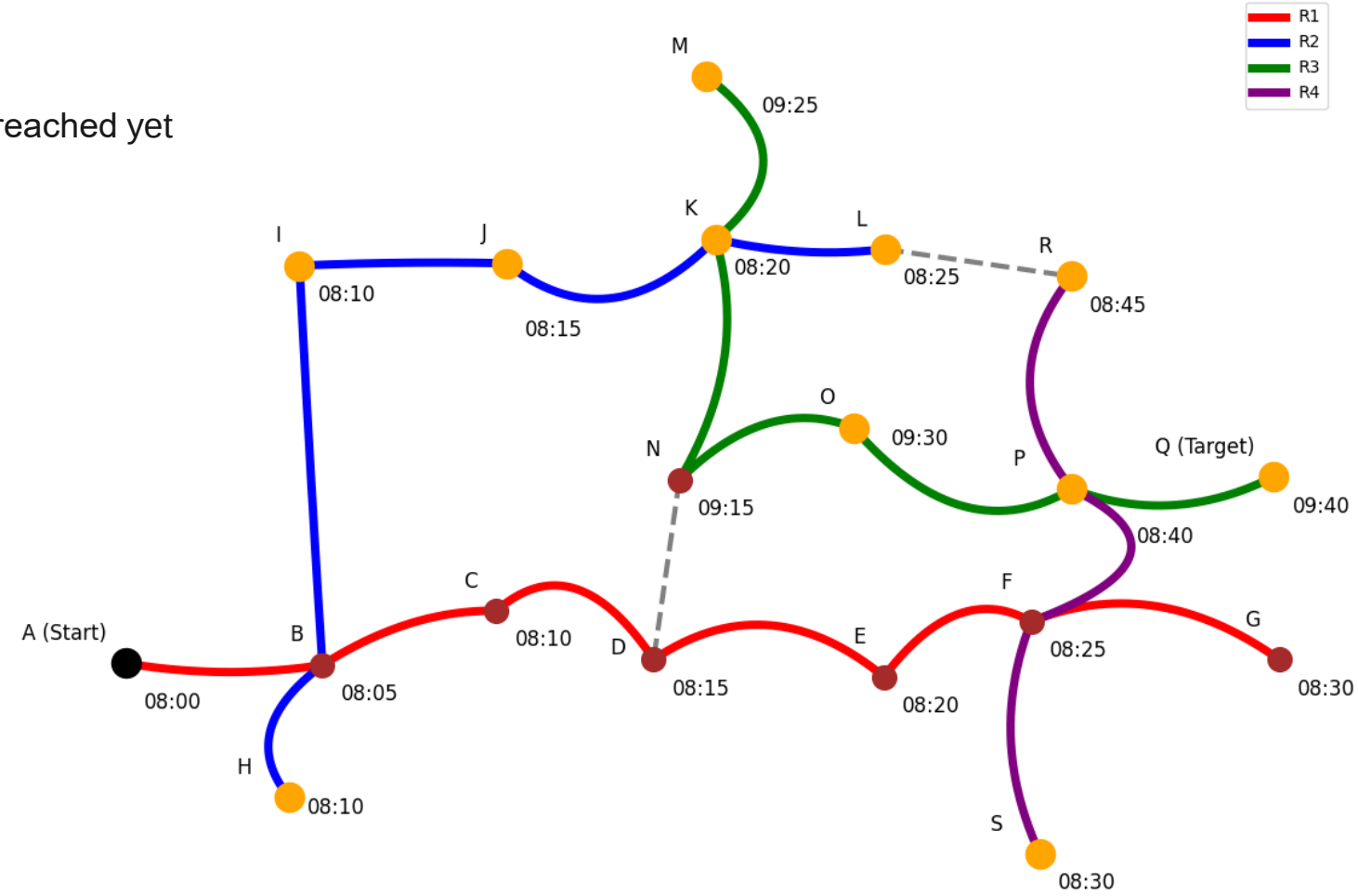
RAPTOR Algorithm (Round 1 – Footpath Relaxation)

- Stop D has a transfer to N which can now be fulfilled



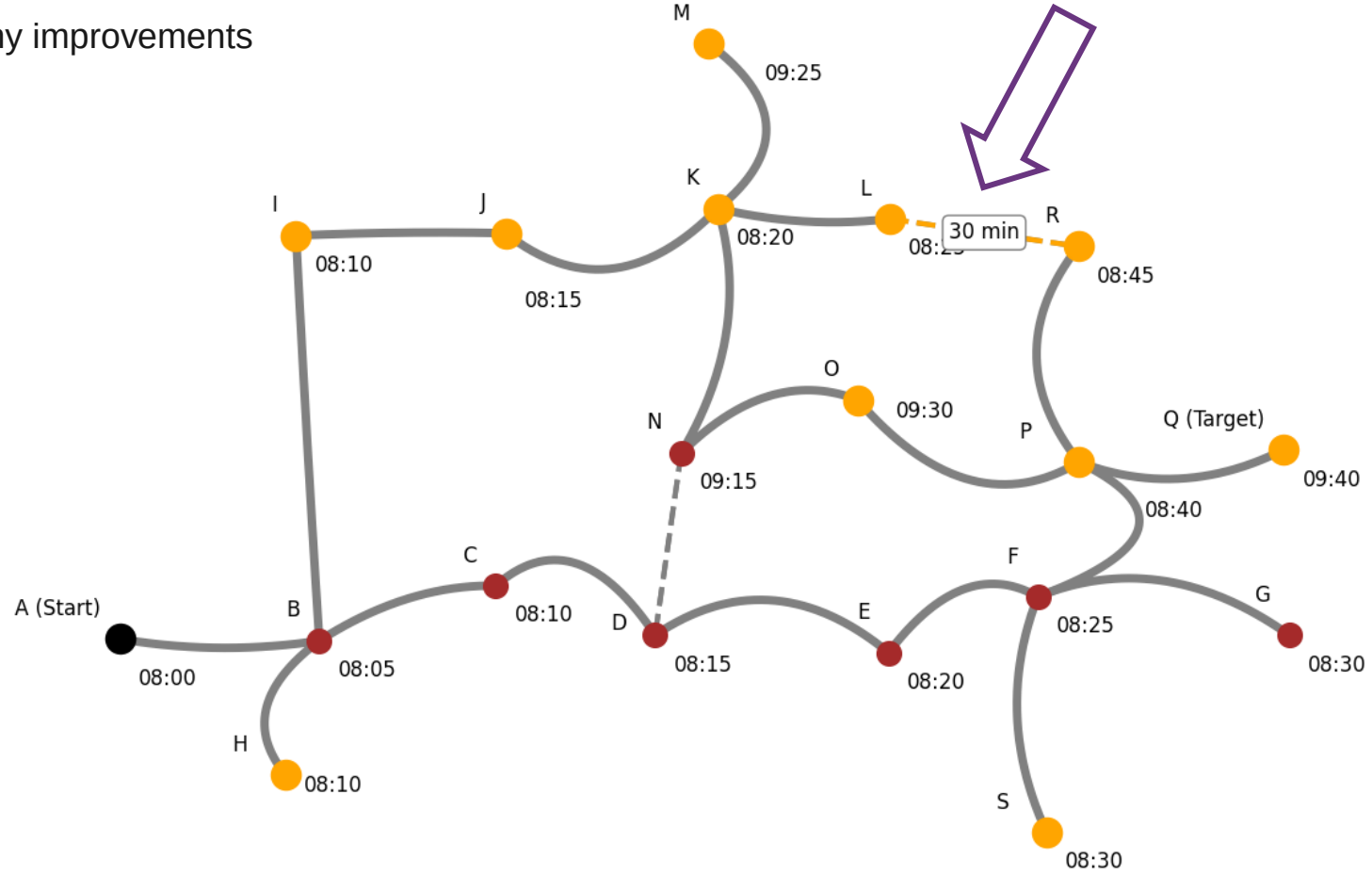
RAPTOR Algorithm (Round 2 – Route Scanning)

- Now all routes need to be scanned
- Marking all stops that haven't been reached yet



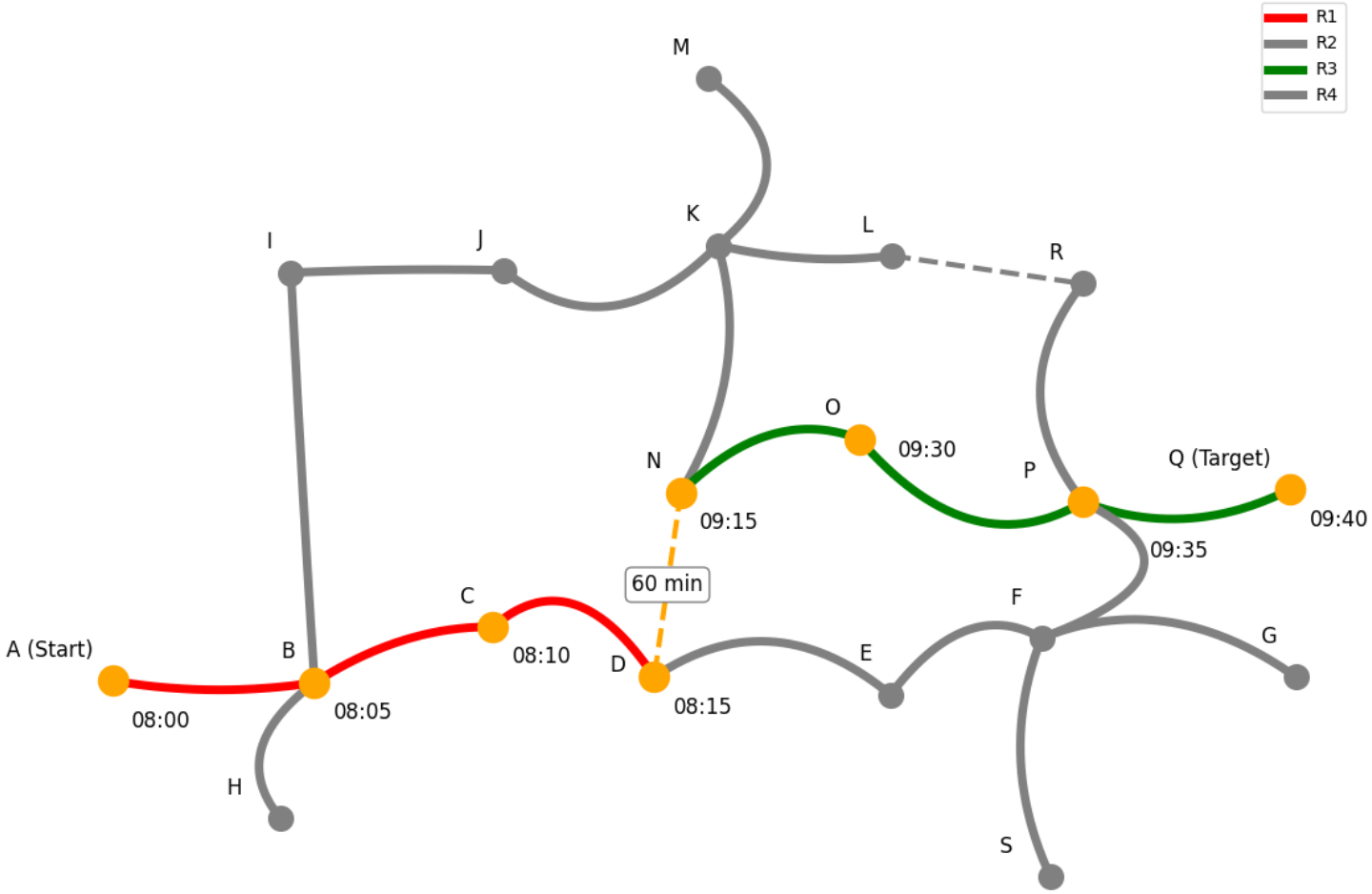
RAPTOR Algorithm (Round 2 – Footpath Relaxation)

- Transfer from L to R does not add any improvements



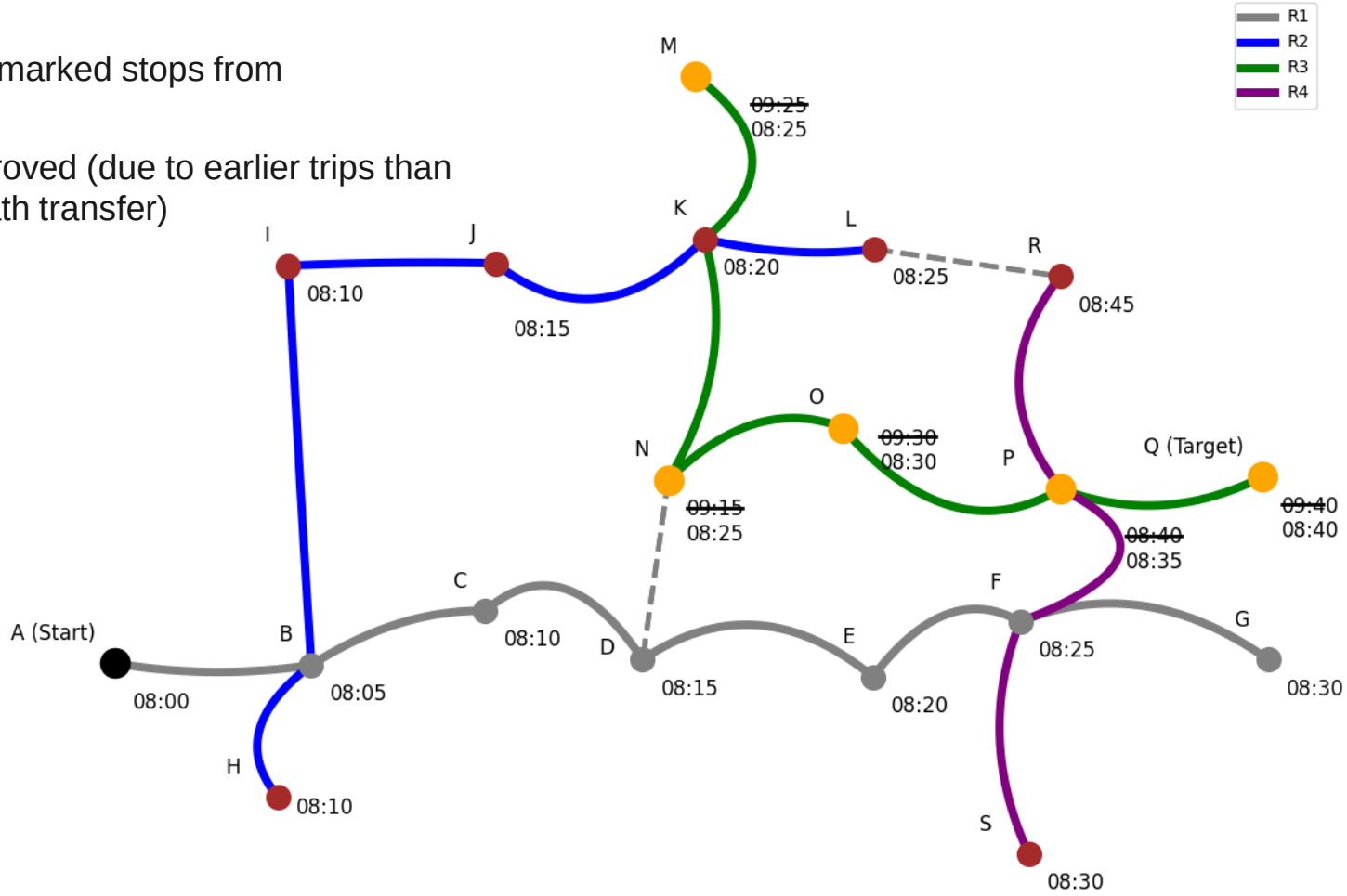
RAPTOR Algorithm

- First Pareto-optimal connection



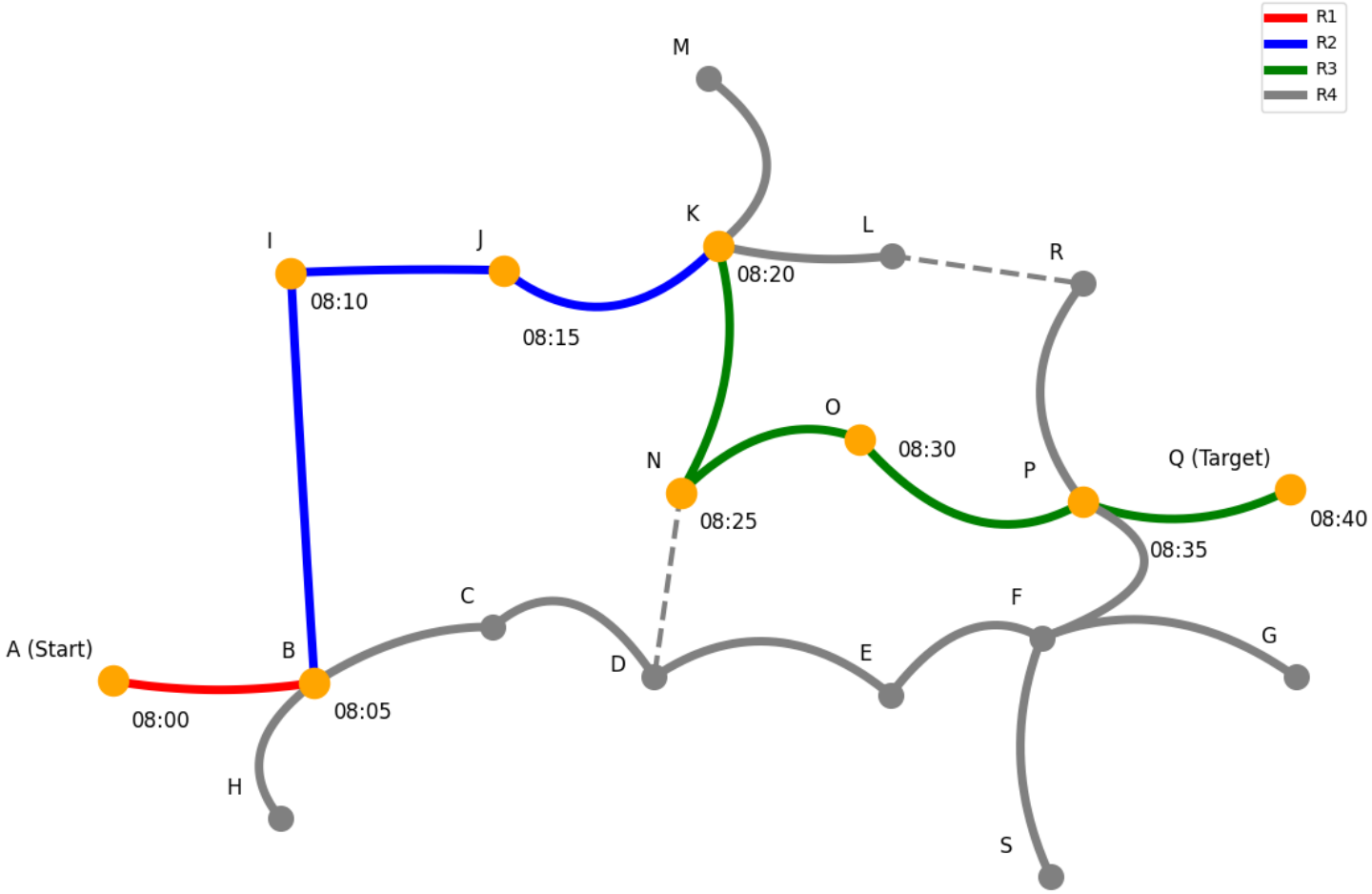
RAPTOR Algorithm (Round 3 – Route Scanning)

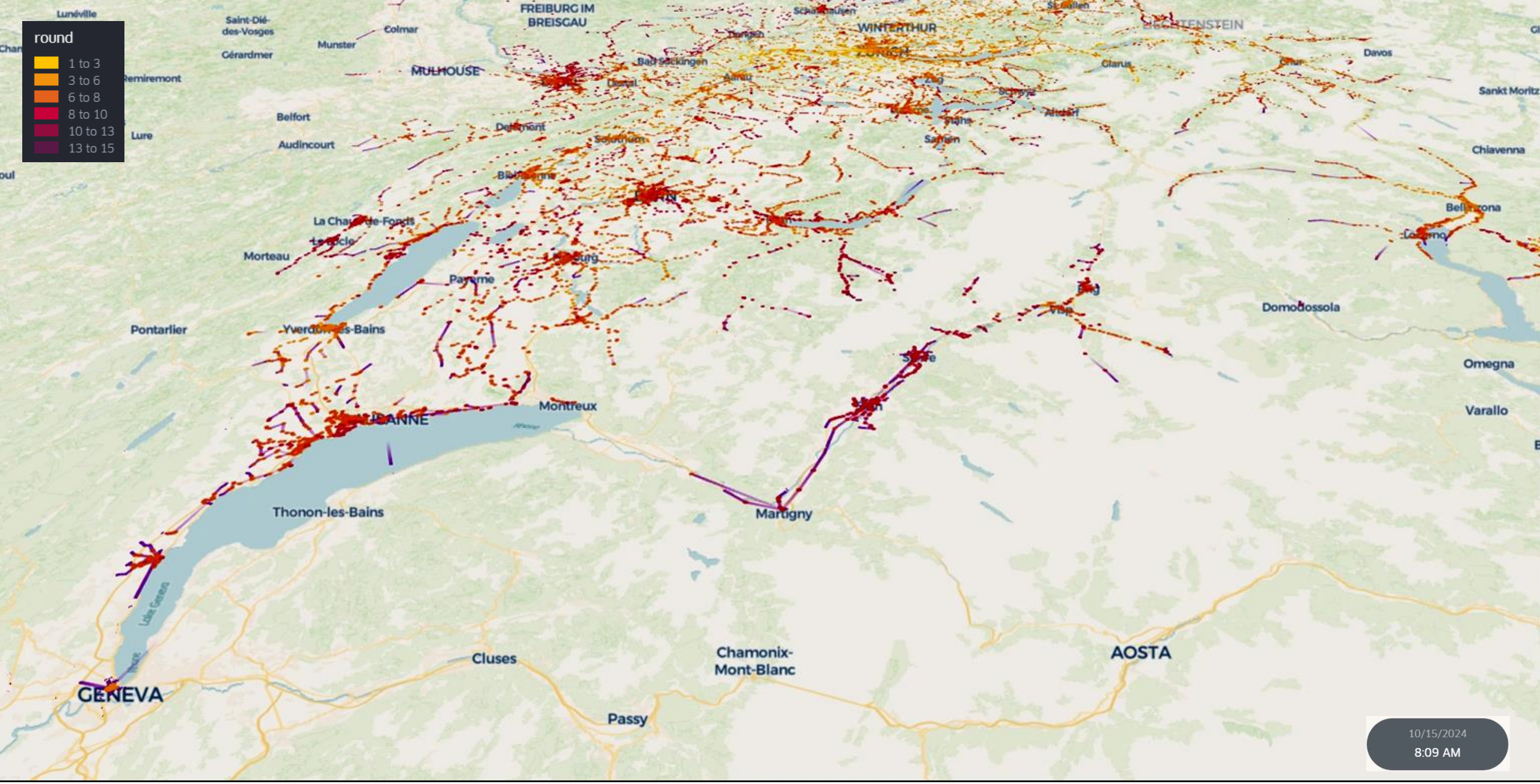
- Scanning all routes passing through marked stops from last round (R2, R3 and R4)
- Only stops on R3 can be further improved (due to earlier trips than previously achievable through footpath transfer)



RAPTOR Algorithm

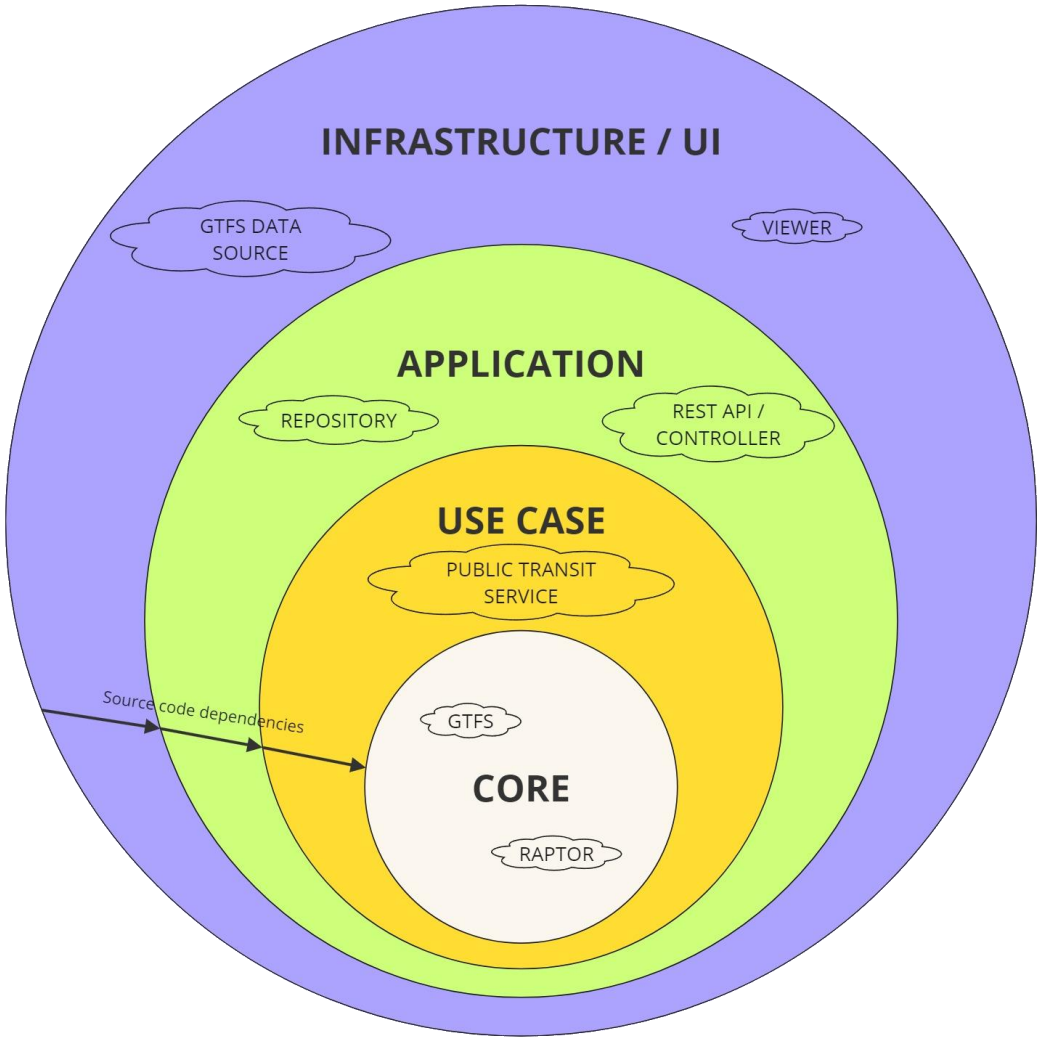
- Second Pareto-optimal connection:
  More rounds but earlier arrival time



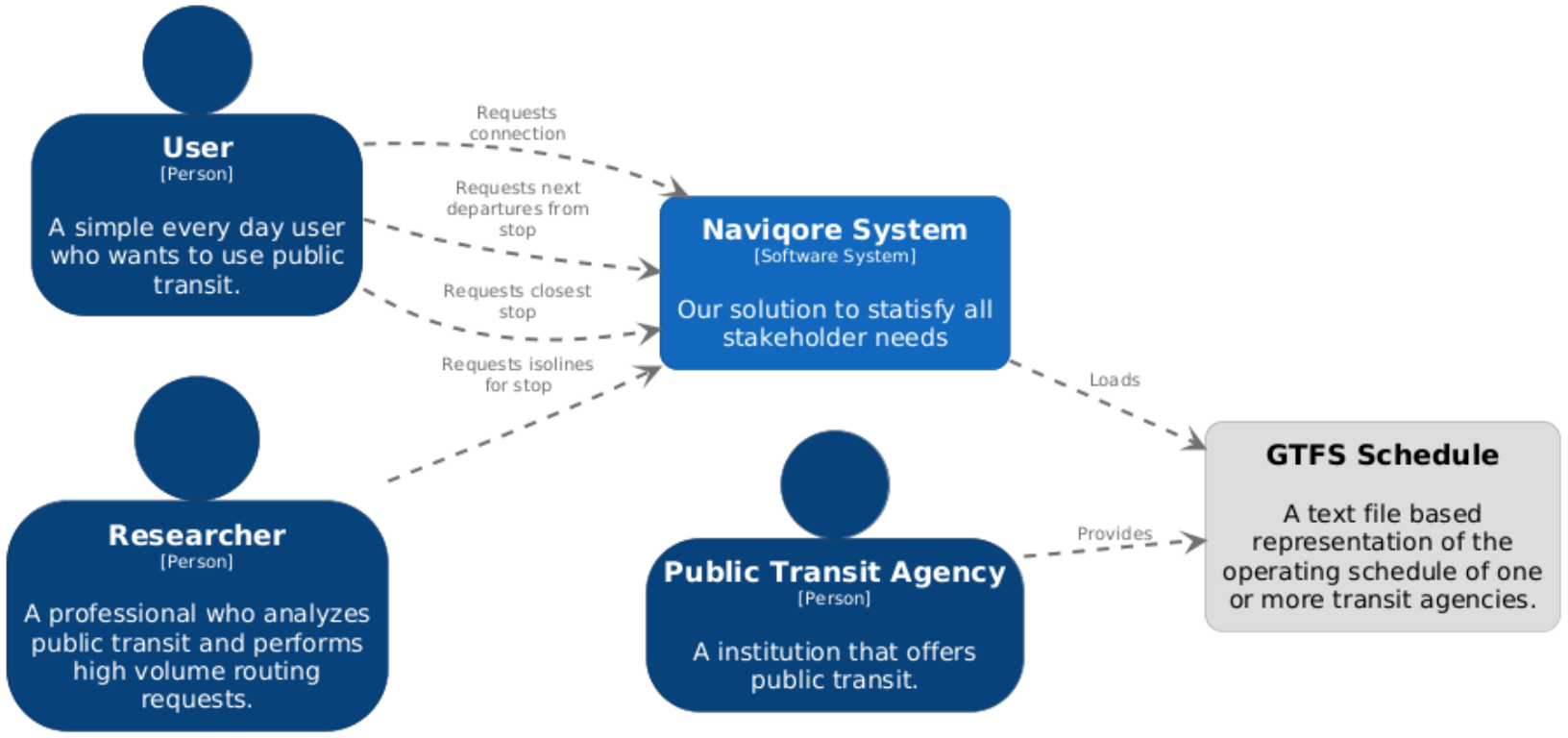

round
1 to 3
3 to 6
6 to 8
8 to 10
10 to 13
13 to 15
Lunéville
Saint-Dié-des-Vosges
Colmar
Munster
Gérardmer
Remiremont
MULHOUSE
FREIBURG IM BREISGAU
Bad Säckingen
WINTERTHUR
Davos
Sankt Moritz
Chiavenna
Belfort
Lure
Audincourt
Delémont
Morteau
La Chaux-de-Fonds
Le Locle
Payerne
Pontarlier
Yverdon-les-Bains
Montreux
Thonon-les-Bains
Martigny
Domodossola
Omegna
Varallo
Cluses
Chamonix-Mont-Blanc
AOSTA
GENEVA
Passy
10/15/2024
8:09 AM

Architecture

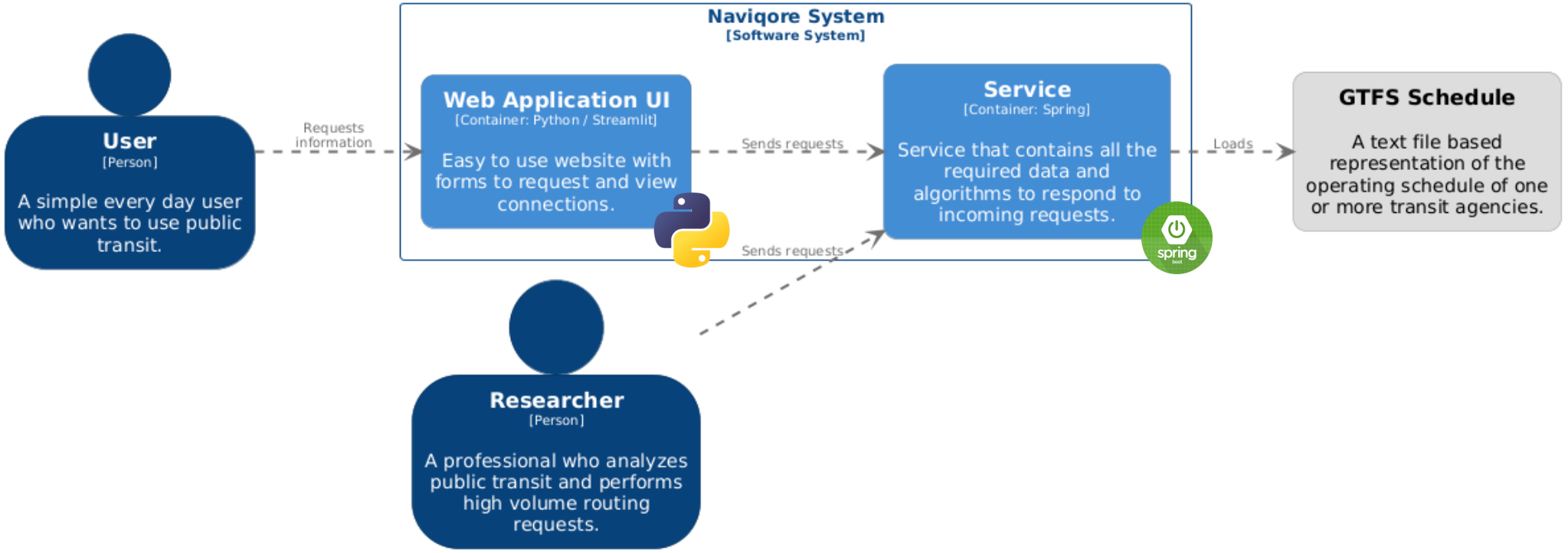
- Clean and **Onion** Architecture.
- Four **layers**
  - Core, Use Case, App, Infrastructure/UI
- Inward dependencies via **dependency injection**
- Inner **abstractions**, outer implementations
- **Loose coupling**, replaceable implementations
- App layer isolates
  - Spring Boot components
  - Repository: Timetable data source



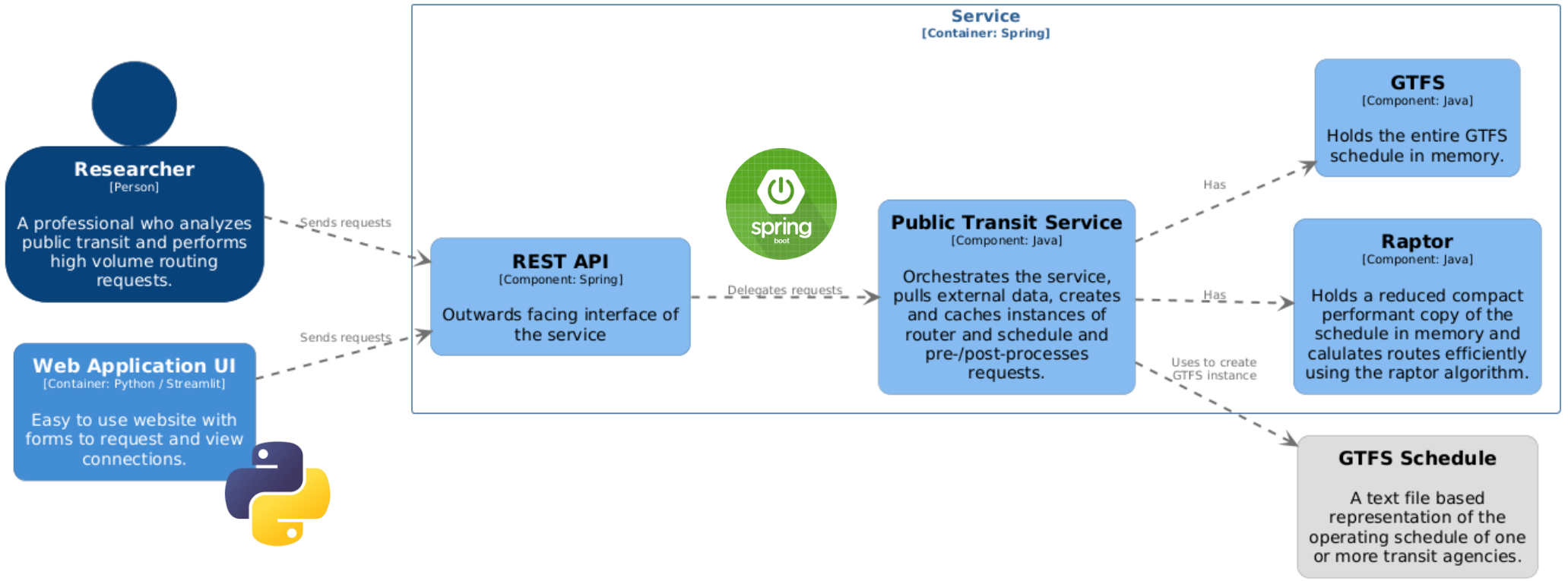
C4 Model – System Context



C4 Model - Containers



C4 Model – Components

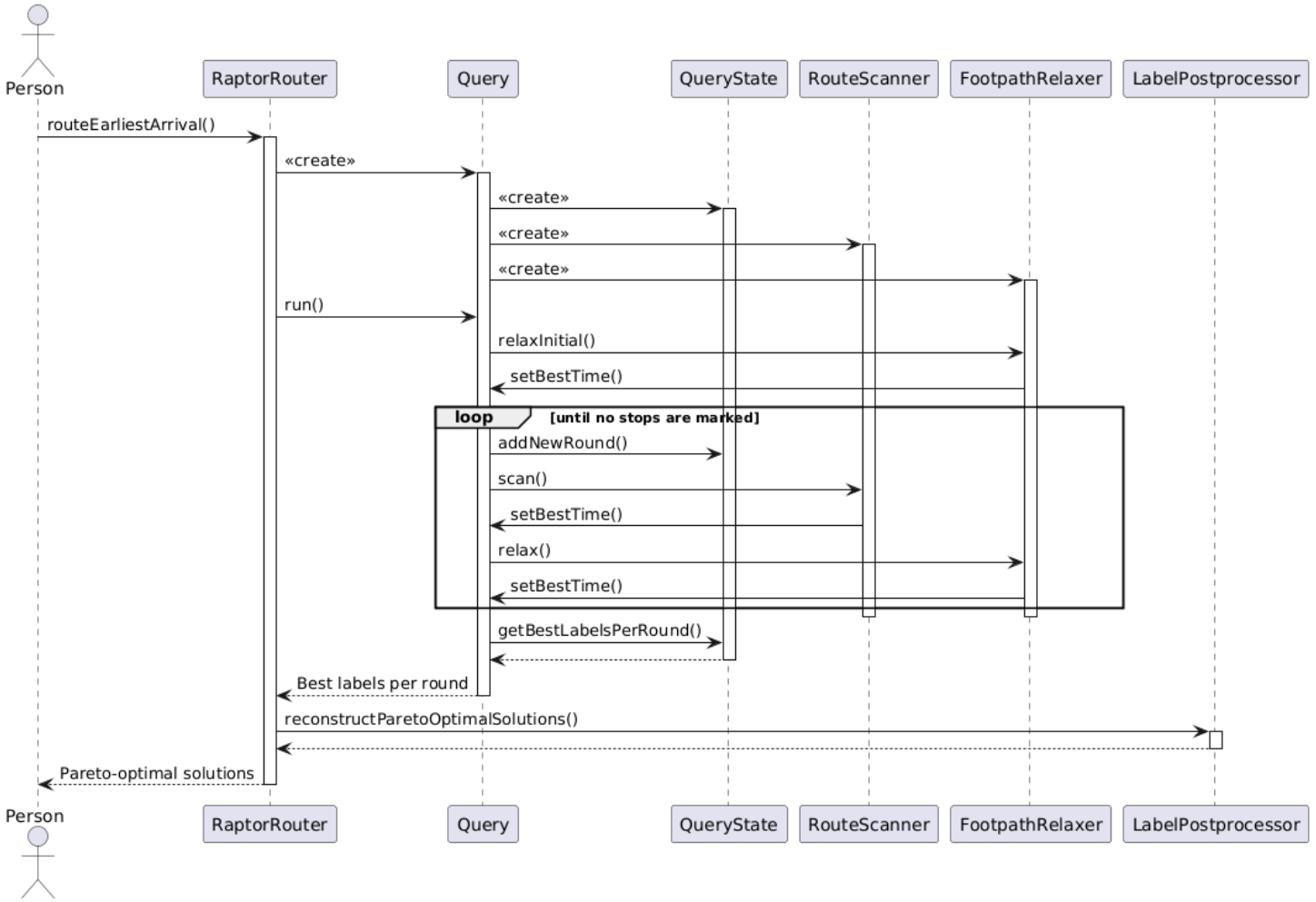


RAPTOR – Extended Features

- **Multi-stop routing:** Routes from multiple origins to multiple destinations, factoring in walking times
- **Reverse routing:** Optimize for latest departure instead of earliest arrival times
- **Multi-day routing:** Supports routing across multiple service days
- **Customizable queries:**
  - Transport mode, max transfers, walking limits
  - Wheelchair accessibility and bike accommodations
- **Range queries:** Find routes with shortest duration within specified departure windows
- **Isolines:** Calculate accessibility within a time range without predefined destinations
- **Performance:** Optimized stop time arrays for better CPU cache efficiency, no hashsets
- **Caching:** LRU eviction cache for masked stop times to enhance performance

Extended RAPTOR – Query Route Calculation

- **Initialization**: Mark first stop
- Initial **footpath relaxation**
- Main Loop: **Round-based search**
  - Add new round
  - **Scan** routes,
    mark improved stops
  - **Relax** footpaths,
    mark improved stops
- **Termination**: End when no improvements possible
- **Reconstruct**: Best route

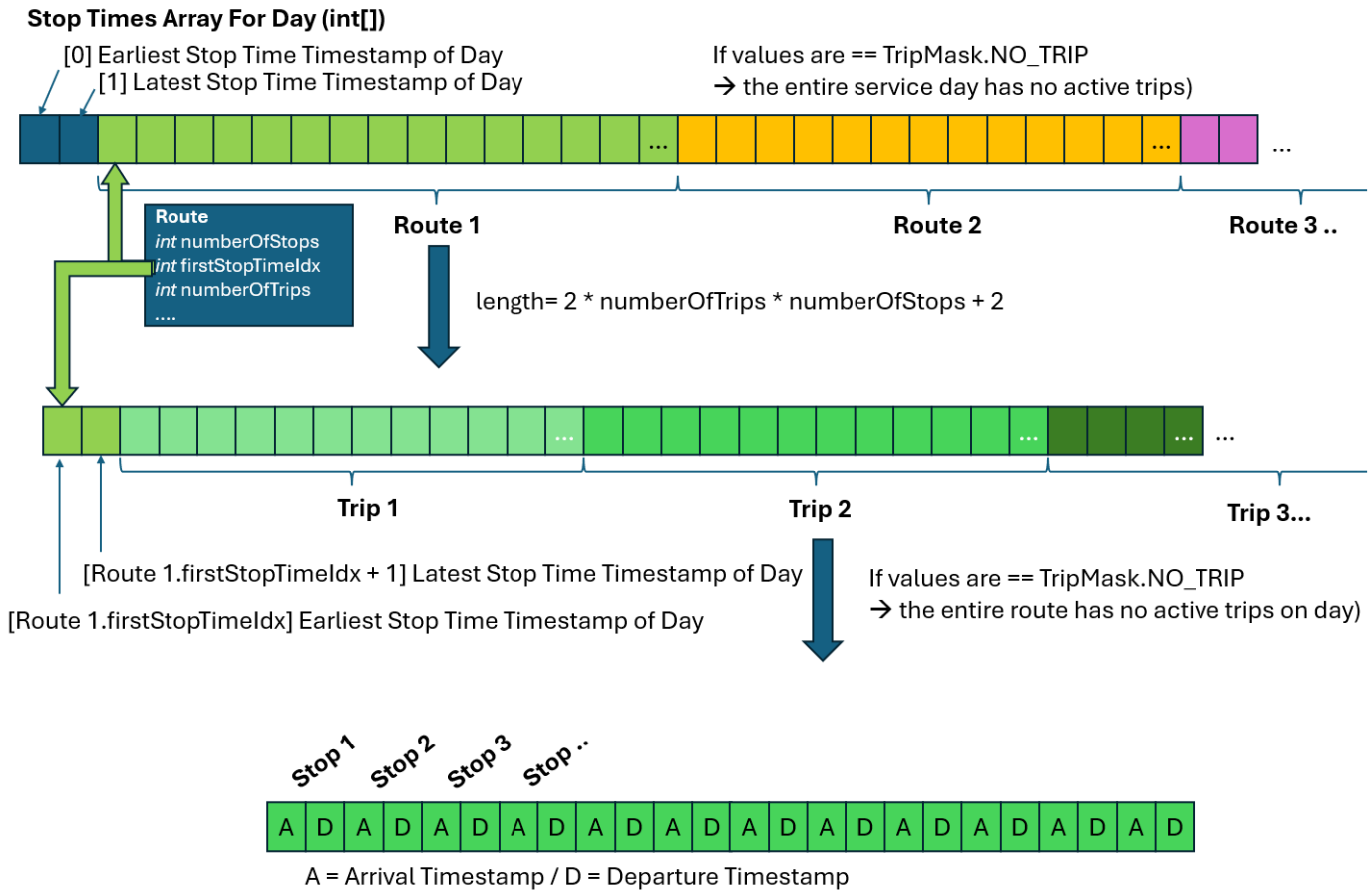


Extended RAPTOR – Stop Times Array

- **Indexing** a stop time:

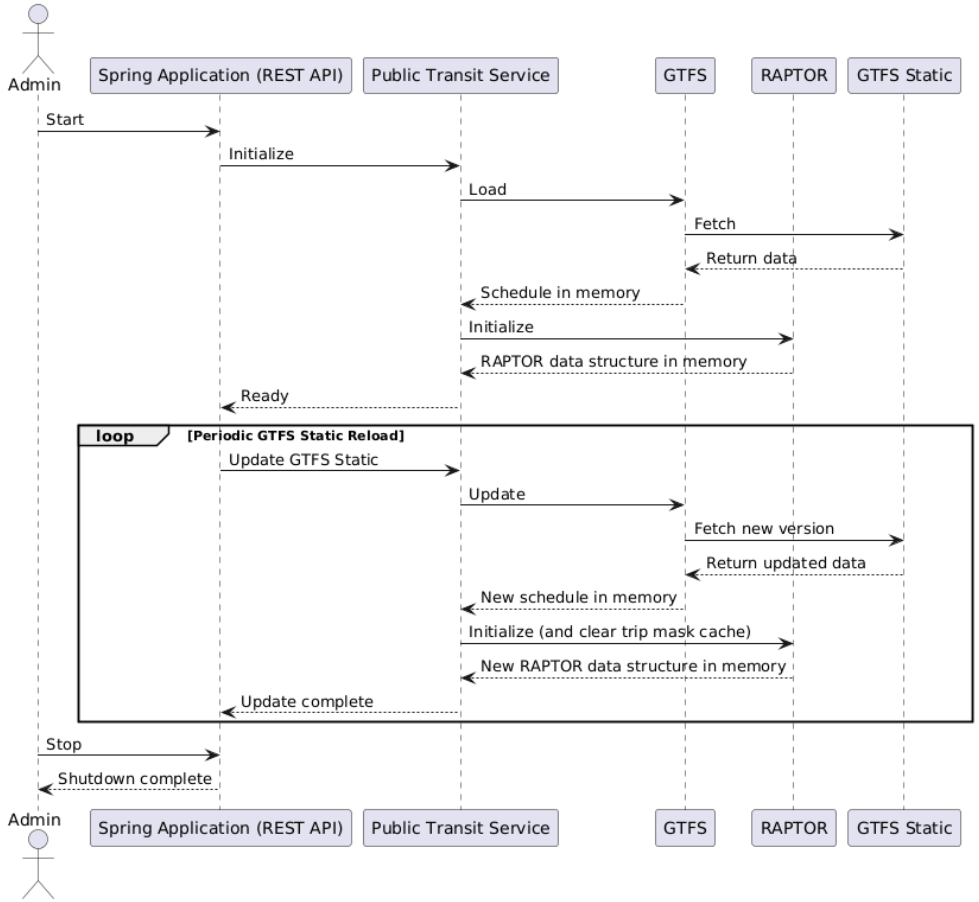
```
int index = route.firstStopTimeIdx
+ 2 + 2 * (tripOffset * numberOfStops)
+ stopOffset + typeConstant
```

  - `tripOffset`: Which trip number should be retrieved (can be incremented for the number of trips from route object)
  - `numberOfStops`: Represents the number of stops each trip has (same for a given route)
  - `stopOffset`: Which stop number the time should be retrieved for
  - `typeConstant`: 0 if arrival time, 1 if departure time
  - +2: Skip earliest and latest trip time of the route
  - *2: Because every stop on the route has two values (arrival/departure)

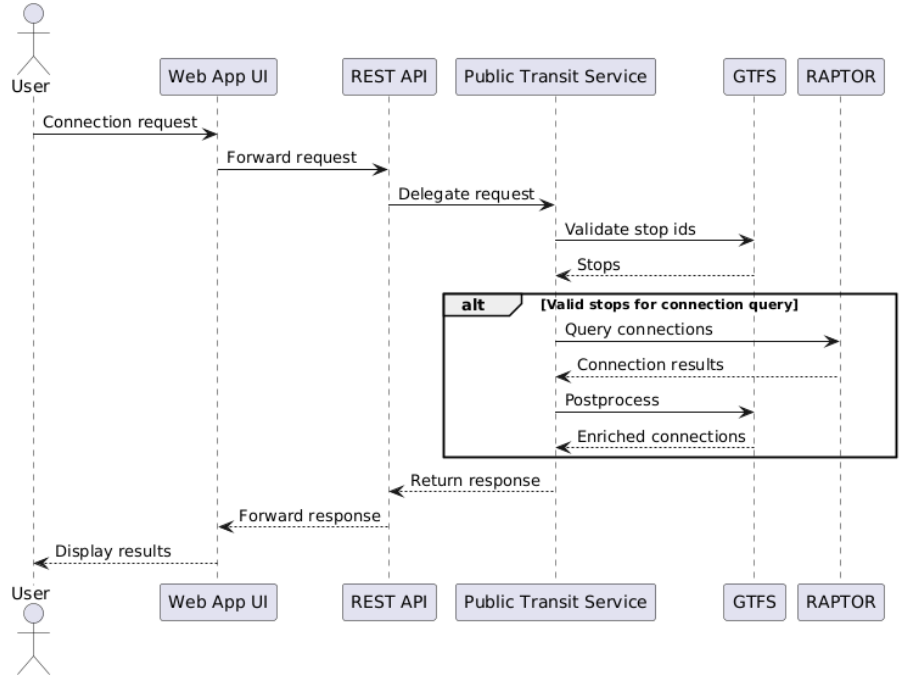


Public Transit (Spring) Service

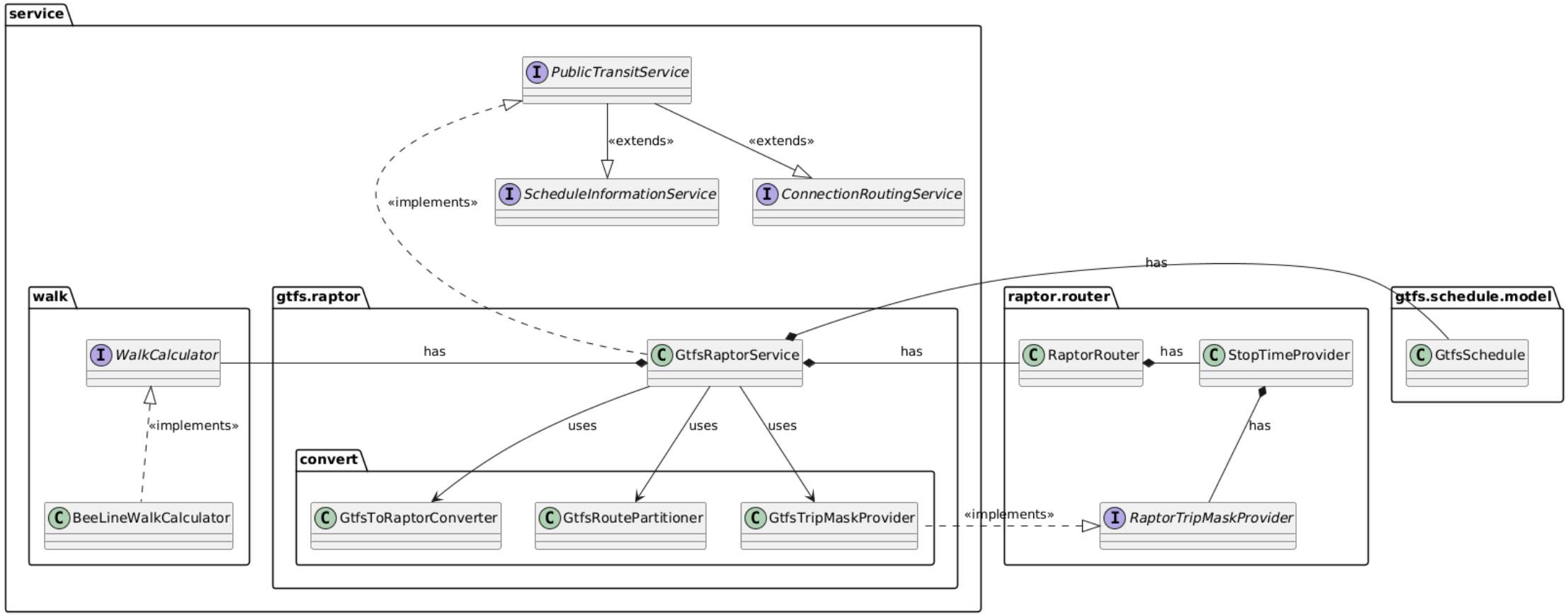
Application startup



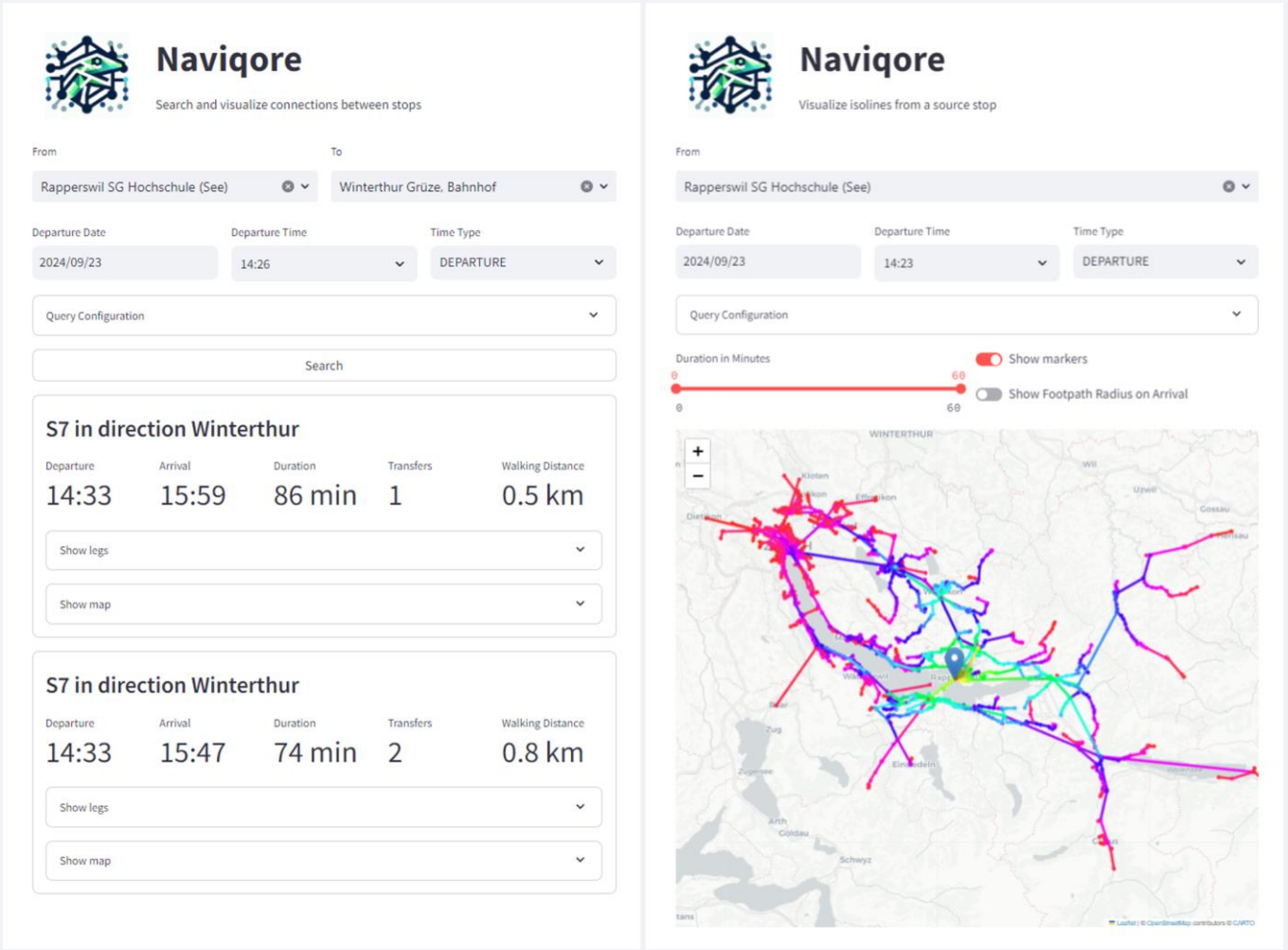
Connection request



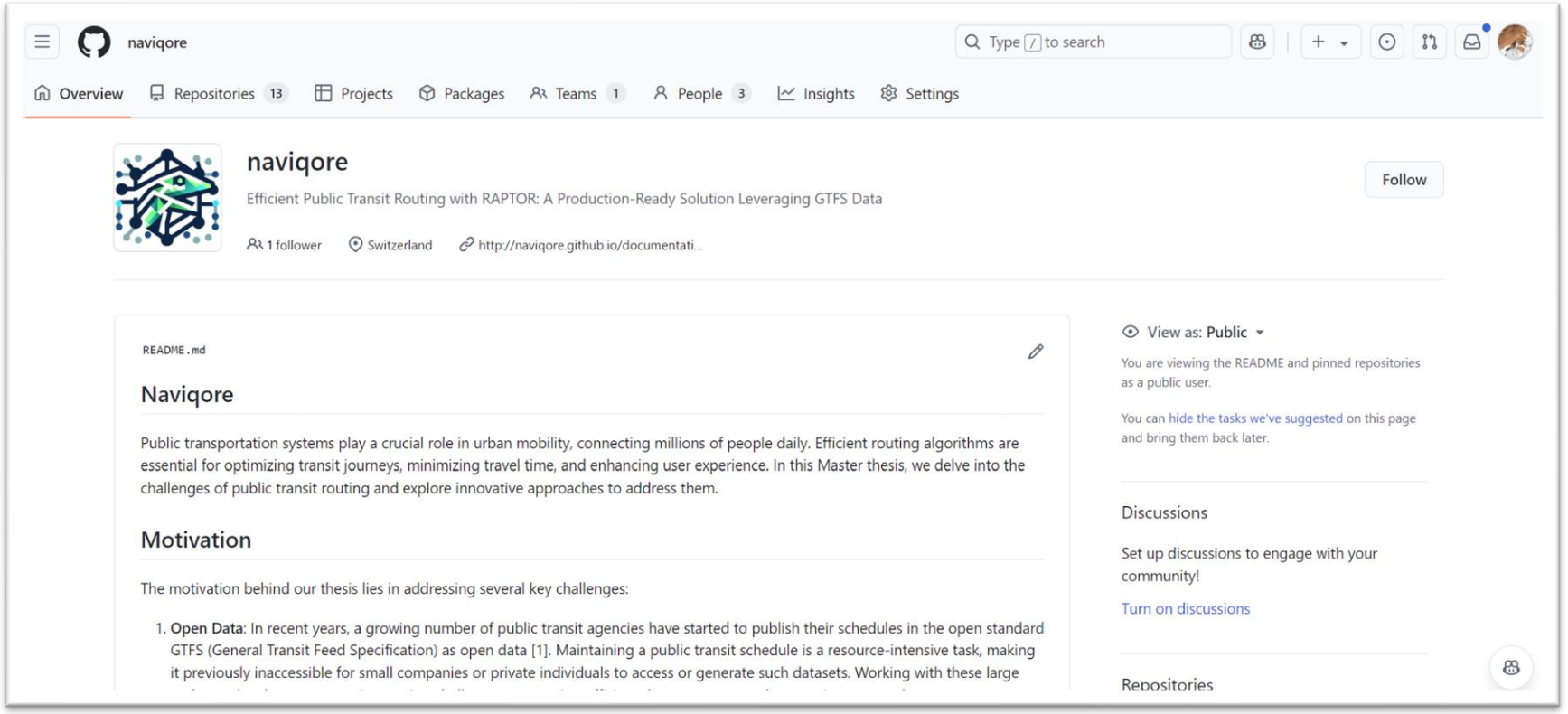
Public Transit (Spring) Service



Demo NAVIQORE



Open-source Software

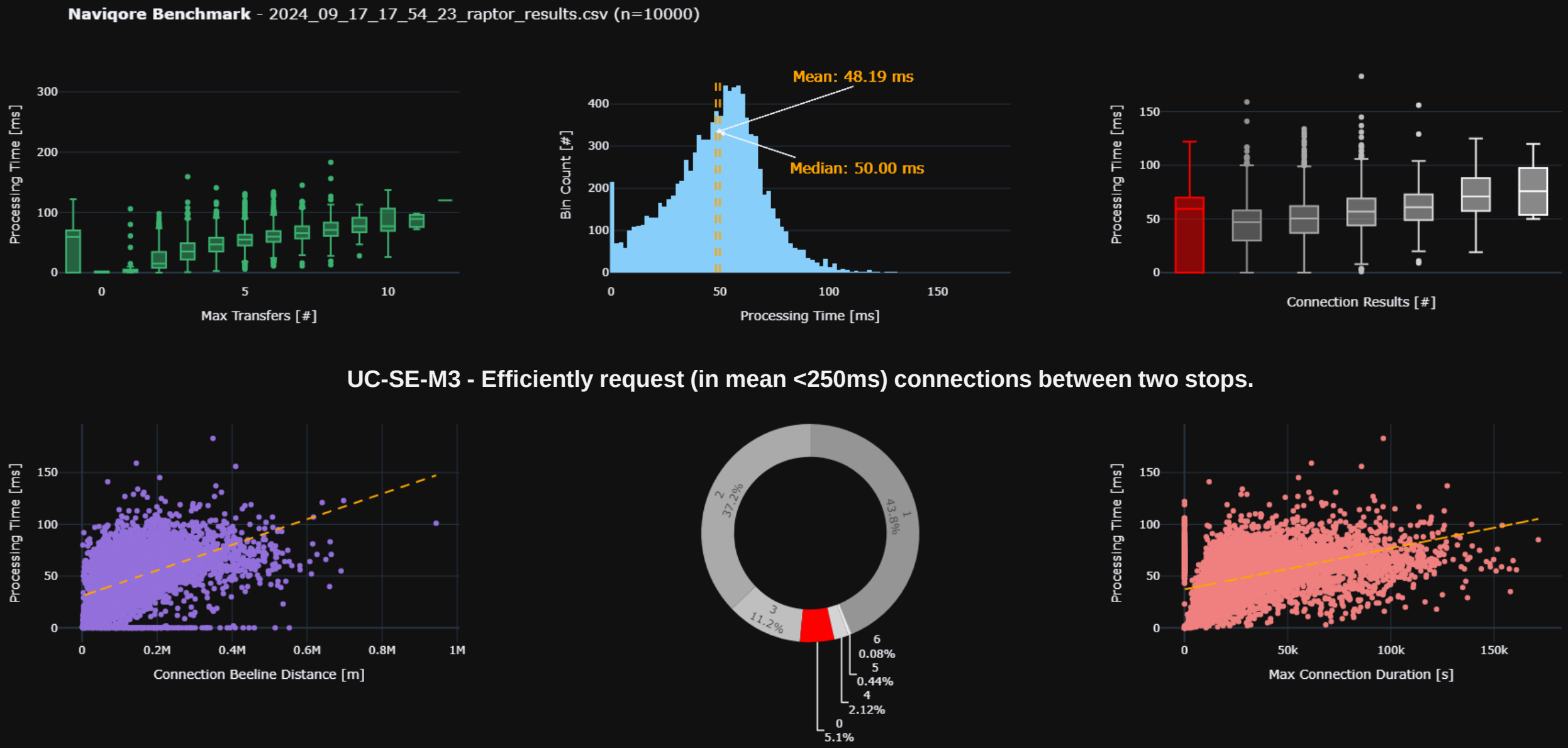


naviqore/public-transit-service

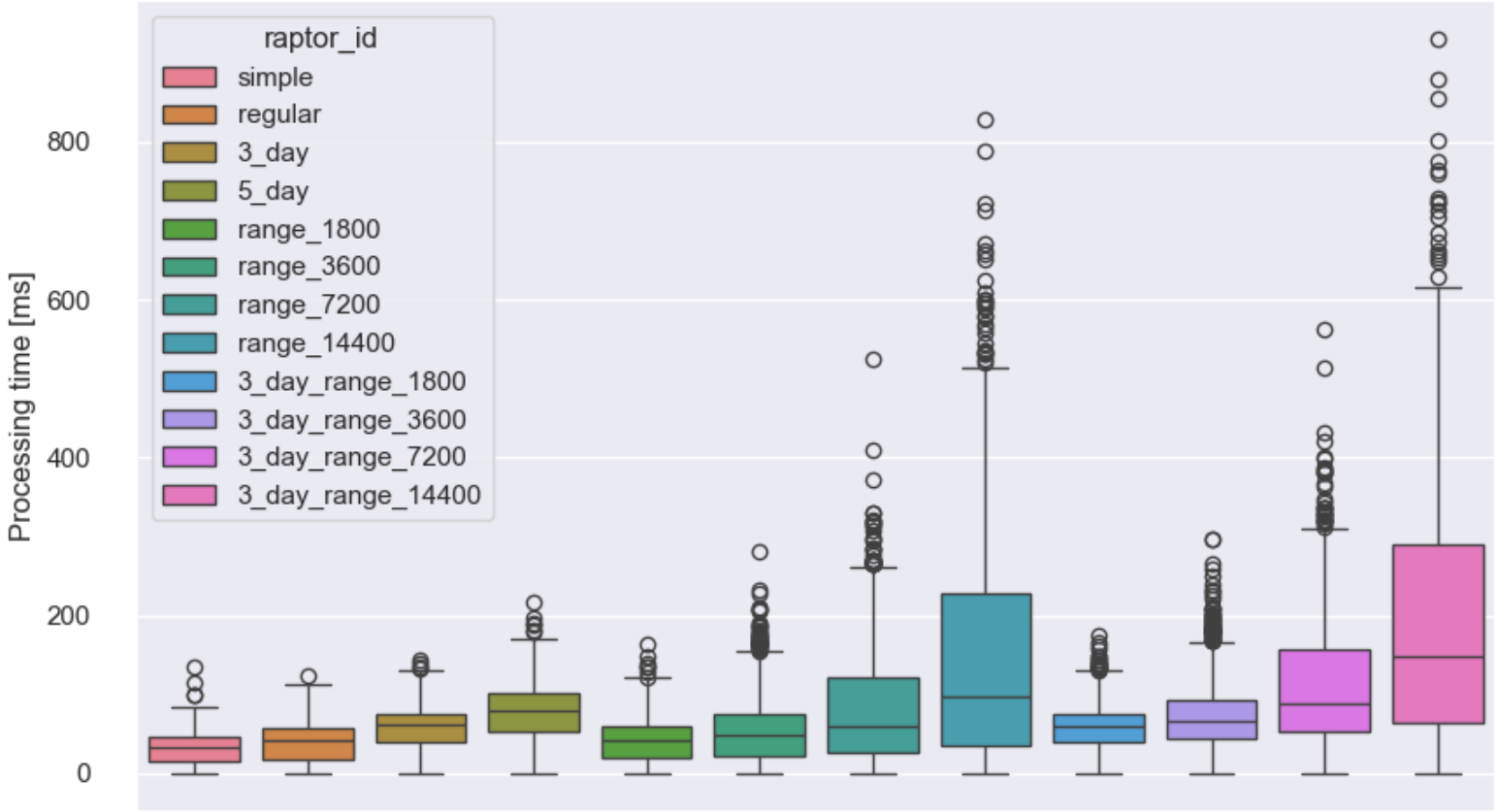
naviqore/public-transit-client

naviqore/public-transit-viewer

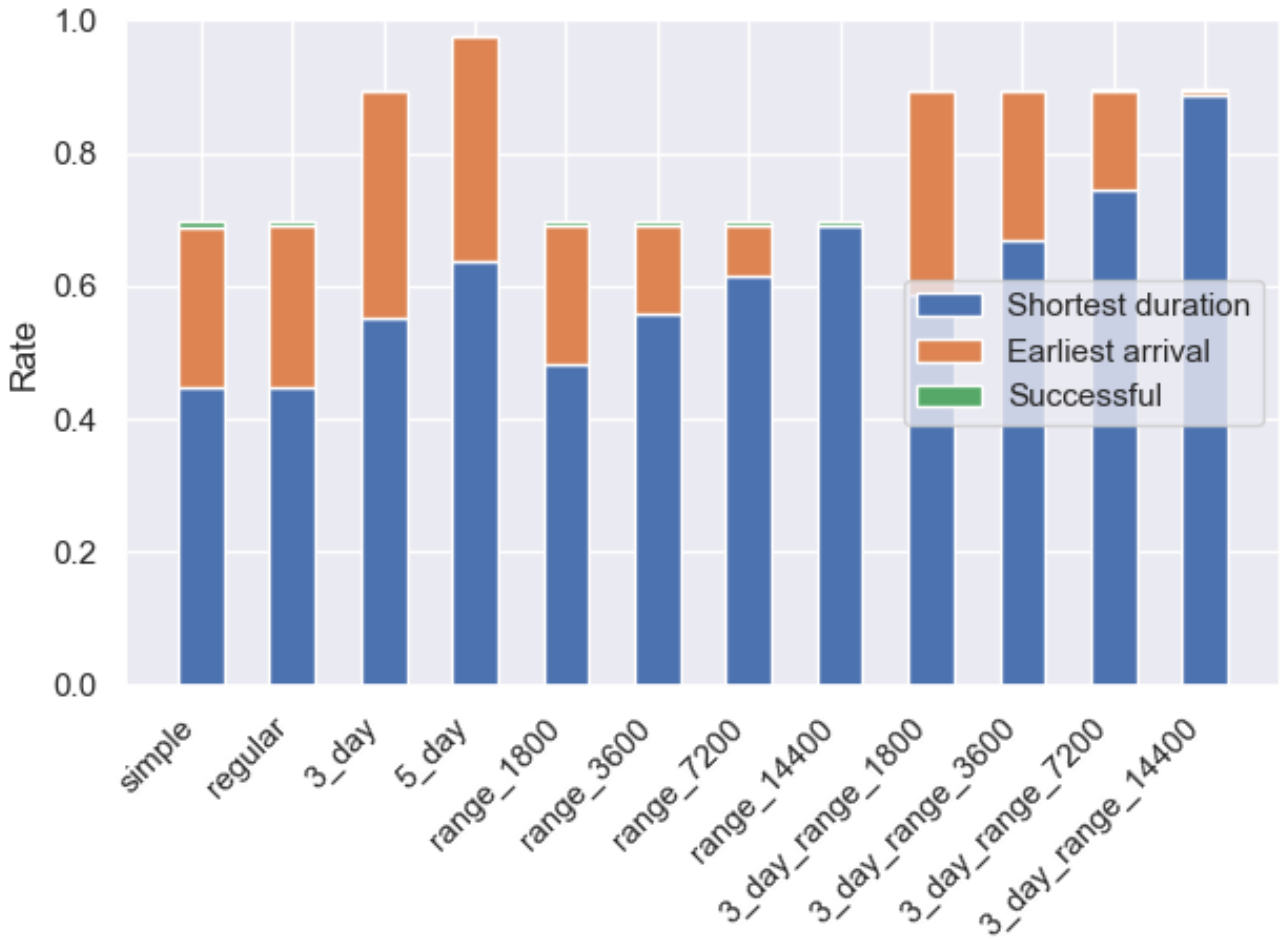
naviqore/raptorxx


Navigore Benchmark - 2024_09_17_17_54_23_raptor_results.csv (n=10000)
Processing Time [ms]
Max Transfers [#]
Mean: 48.19 ms
Median: 50.00 ms
Bin Count [#]
Processing Time [ms]
Processing Time [ms]
Connection Results [#]
UC-SE-M3 - Efficiently request (in mean <250ms) connections between two stops.
Processing Time [ms]
Connection Beeline Distance [m]
2
37.2%
1
43.8%
3
11.2%
6
0.08%
5
0.44%
4
2.12%
0
5.1%
Processing Time [ms]
Max Connection Duration [s]

Benchmark – Java RAPTOR: Performance

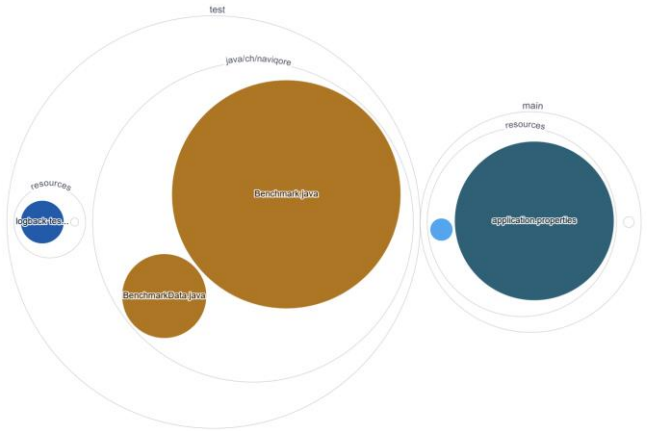


Benchmark – Java RAPTOR: Success Rate



Code Metrics

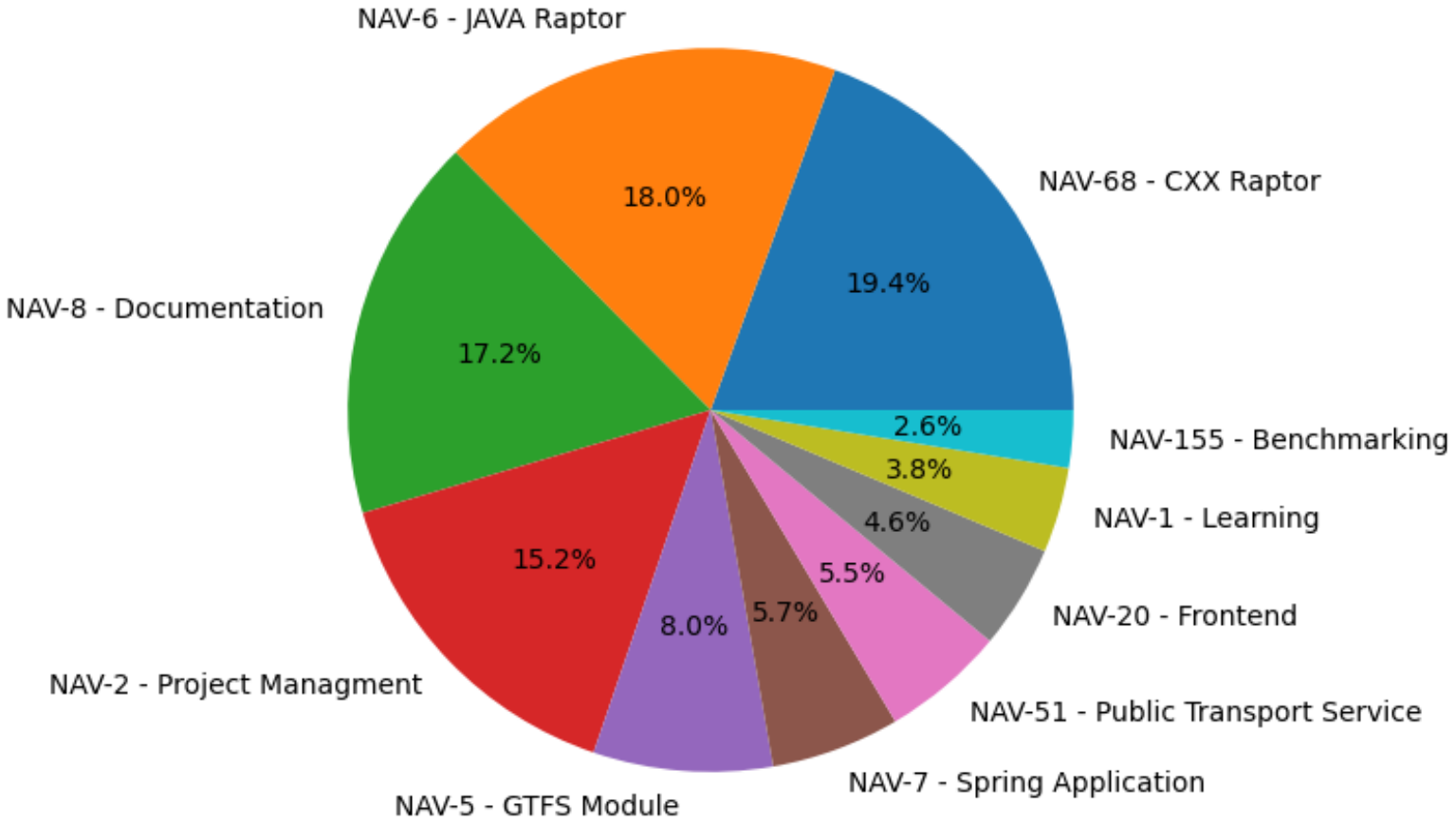
- **Coding**: ~ 1300 commits, ~ 250 pull requests
- **Testing**: ~ 650 test cases, ~ 85% coverage



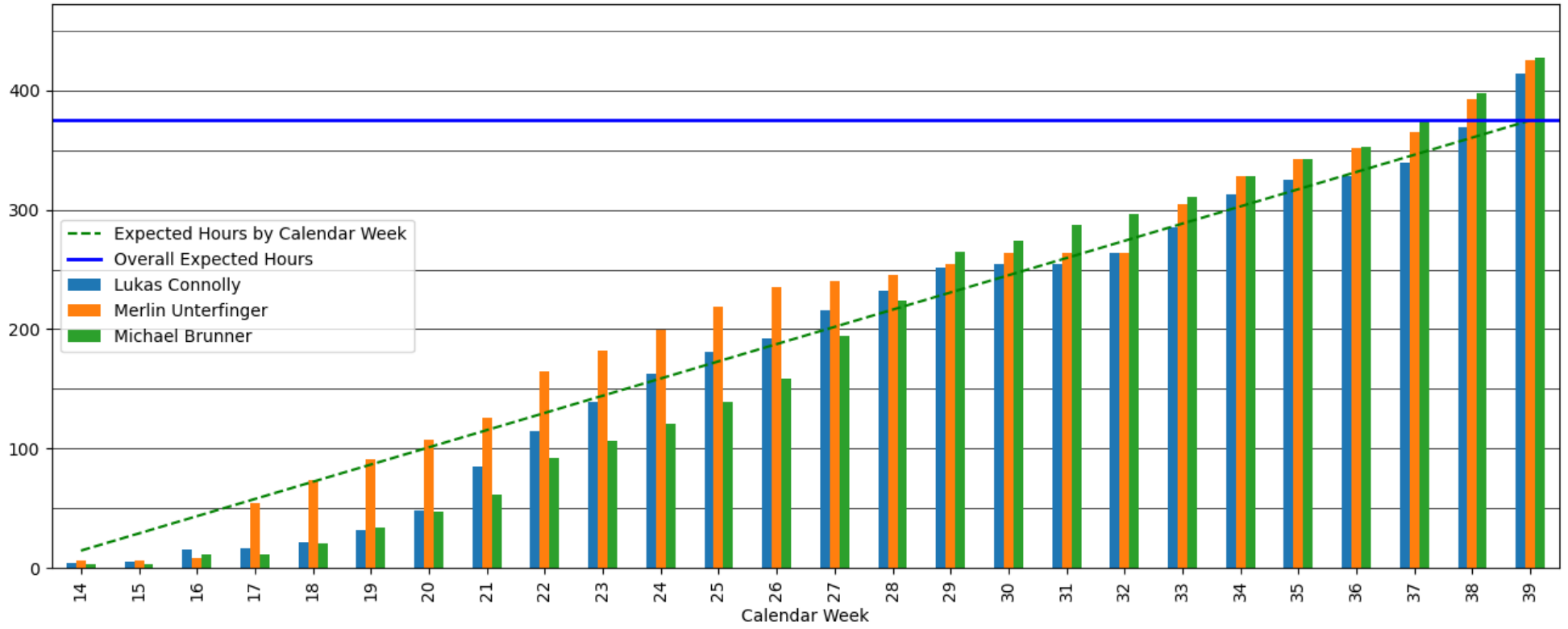
Repository Name	Language	Commits	Lines of Code *(Test)*	Test Cases *(Integration)*	Test Coverage
public-transit-service	Java	751	12116 *(5662)*	569 *(51)*	86%
public-transit-client	Python	72	822 *(419)*	23 *(15)*	80%
public-transit-viewer	Python	125	1007 *(21)*	1 *(1)*	-
raptorxx	C++	264	9164 *(1018)*	47 *(0)*	-

- **Notes:**
  - No coverage for **public-transit-viewer** due to Streamlit subprocess
  - No coverage for **raptorxx** due to MSVC and CMake limitations

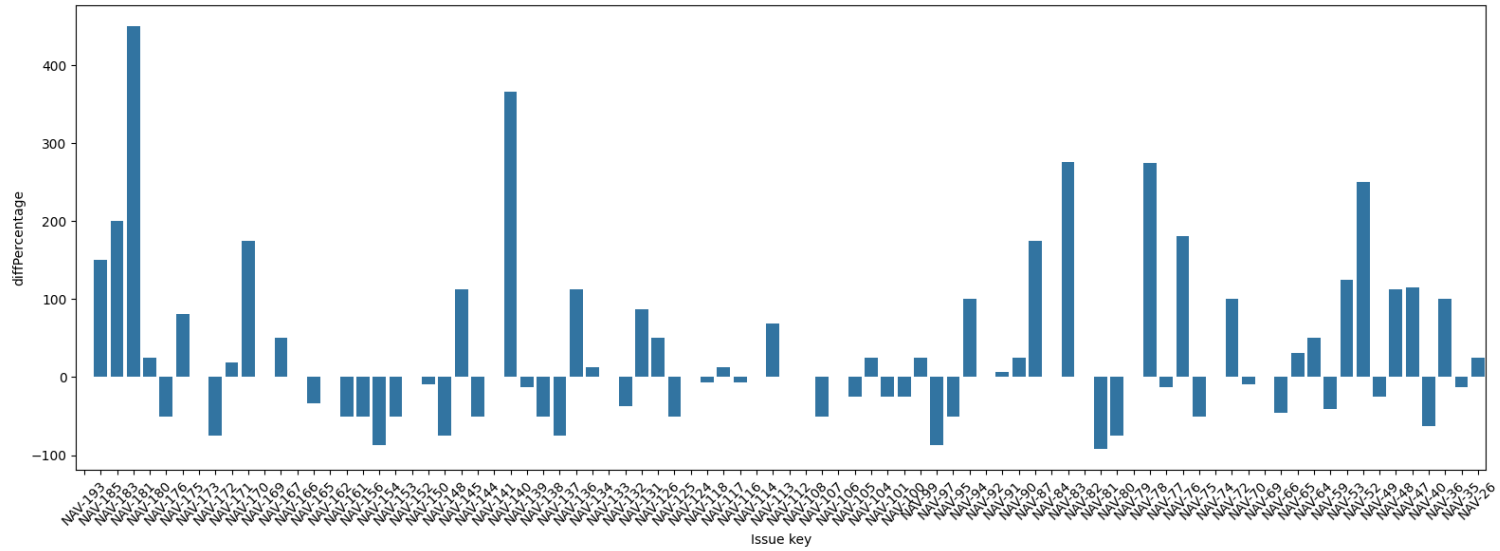
Time Tracking – Epics

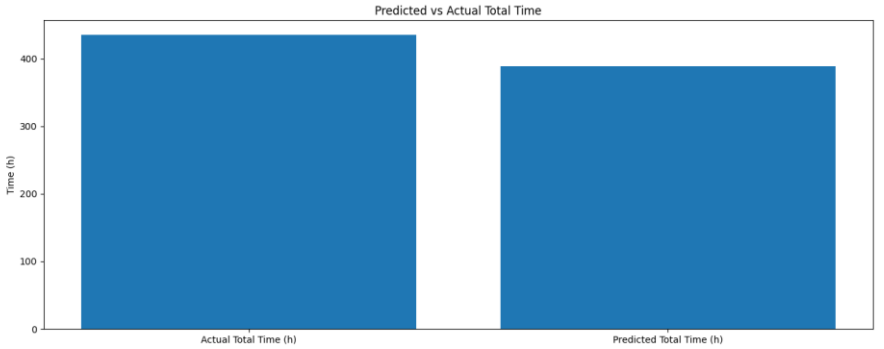


Time Tracking – Developers



Time Tracking – Estimates





C++ vs. Java – Simple RAPTOR

- Motivation: Performance comparison
- Challenges:
  - CMake (build system)
  - Different OS / compiler
  - Memory management
- **Results:**
  - Short distances slower
  - Long distances faster
- Our Java implementation uses primitive array, C++ dynamic `std::vector`

Source Stop	Target Stop	Iterations	Java Elapsed Time (ms)	C++ Elapsed Time (ms)	Average Difference (ms)	Average Difference (%)
8589640 (St. Gallen, Vonwil)	8579885 (Mels, Bahnhof)	100	13	19	-6	-31.6%
8574563 (Maienfeld, Bahnhof)	8587276 (Biel/Bienne, Taubenloch)	100	55	38	17	44.7%
8588524 (Sion, Hôpital Sud)	8508896 (Stans, Bahnhof)	100	45	20	25	125.0%
8510709 (Lugano, Via Domenico Fontana)	8579255 (Lausanne, Pont-de-Chailly)	100	46	29	17	58.6%
8574848 (Davos Dorf, Bahnhof)	8576079 (Rapperswil SG, Sonnenhof)	100	12	15	-3	-20.0%

Table 6.4: Comparison of Java and C++ performance.

High-performance C++ Code → A Double-edged Sword

- **Secure programming vs. performance?**
- Raw pointer is **1.51 x faster** than shared pointer
- Raw array is
  - **3.09 x faster**
  - with `reserve(SIZE)` **1.41 x faster**

  than **`std::vector`** for adding/getting items

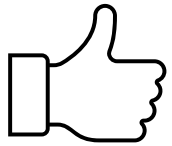
```
------------------------------------------------------------------
Benchmark                        Time             CPU   Iterations
------------------------------------------------------------------
BM_raw_reference         367484 ns       325239 ns         2306
BM_shared_reference      495431 ns       452080 ns         1659
BM_vector                 88914 ns        78125 ns        11200
BM_raw_array             125377 ns       112997 ns         4978
```

```cpp
constexpr int SIZE = 10'000;

static void BM_raw_reference(benchmark::State &state)
{
    for (auto _ : state) {
        auto vec = std::vector<Song *>();
        vec.reserve(SIZE);

        std::ranges::transform(std::ranges::views::iota(0, SIZE),
                               std::back_inserter(vec),
                               [](int)
                               {
                                   return new Song{"artist", "title"};
                               });

        std::ranges::for_each(vec, [](const auto *f) { delete f; });
    }
}

static void BM_shared_reference(benchmark::State &state)
{
    for (auto _ : state) {
        auto vec = std::vector<std::shared_ptr<Song> >();
        vec.reserve(SIZE);

        std::ranges::transform(std::ranges::views::iota(0, SIZE),
                               std::back_inserter(vec),
                               [](int)
                               {
                                   return std::make_shared<Song>("artist",
"title");
                               });

        vec.clear();
    }
}

BENCHMARK(BM_raw_reference);
BENCHMARK(BM_shared_reference);
```

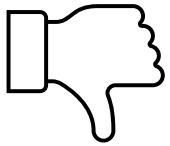
Limitations

- GTFS publications often lack information on **accessibility, bike allowance**, and pathways
- **International routing**: Combining multiple GTFS publications (duplicates?, time zones)
- Current **footpath routing**: Approximation using beeline distance and factor
- Trade-off: **Memory vs. performance** (caching)
- RAPTOR and **C++**: FFM API complexity, multiple OS, …

Conclusion

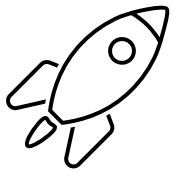


- Final product, **open source**
- Cool **team** and topic
- Software development process
  - Efficiency and **reviews**
- **Knowledge** gain
  - Data structures & algorithms, language specifics, architecture



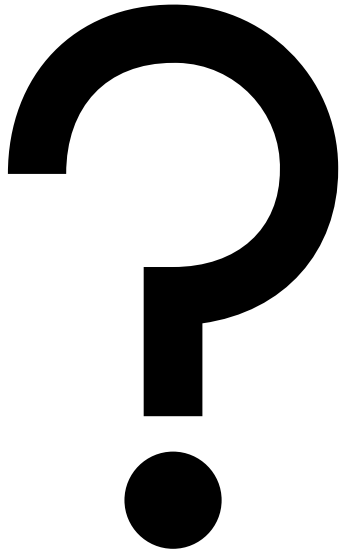
- Little overlap of C++ and Java part of the project
- Work-life balance:
  Strain on personal life…

Outlook



- Support **multi-criteria queries**, *mcRAPTOR*: Fares, journey attractivity, …
- Incorporate **GTFS Realtime**
  - Consume protocol buffers
  - Update GTFS schedule and invalidate cache
- Implement **A*** algorithm for **routing footpaths**
  - Use OSM data to build the graph
  - Additionally: Calculate shapes of transit legs
- **Decoupling** of service and schedule / RAPTOR data
  - Common storage layer: Database holding GTFS and potentially RAPTOR data
  - Improves horizontal scaling

Any Questions



References

[1] "Gtfs cookbook," https://opentransportdata.swiss/de/cookbook/gtfs, n.d., retrieved May 25, 2024.

[2] M. Unterfinger, M. Brunner, and L. Connolly, "Efficient public transit routing with raptor: A production-ready solution leveraging gtfs data," https://github.com/naviqore, 2024, retrieved September 26, 2024.

[3] "General transit feed specification," https://gtfs.org, n.d., retrieved May 25, 2024.

[4] D. Delling, T. Pajor, and R. F. Werneck, "Round-based public transit routing," in *Proceedings of the Meeting on Algorithm Engineering and Experiments (ALENEX)*. SIAM, 2012, pp. 130–140. [Online]. Available: https://doi.org/10.1137/1.9781611972924.13

[5] K. Schwaber and J. Sutherland, "The scrum guide: The definitive guide to scrum: The rules of the game," https://scrumguides.org/docs/scrumguide/v2020/2020-Scrum-Guide-US.pdf, 2020, retrieved May 25, 2024.

[6] J. Sauer, "Public transit journey planning," Lecture slides, Karlsruhe Institute of Technology (KIT), 2021, retrieved May 21, 2024.

[7] R. C. Martin, *Clean Architecture: A Craftsman's Guide to Software Structure and Design*, 1st ed. USA: Prentice Hall Press, 2017. [Online]. Available: https://doi.org/10.5555/3175742

[8] J. Palermo, "The onion architecture, part 1," https://jeffreypalermo.com/2008/07/the-onion-architecture-part-1, 2008, retrieved June 8, 2024.

References

[9] "Spring boot reference documentation," https://spring.io/projects/spring-boot, n.d., retrieved May 31, 2024.

[10] S. Brown, "The c4 model for visualising software architecture," https://c4model.com, n.d., retrieved June 14, 2024.

[11] ——, "The c4 model for software architecture," https://www.infoq.com/articles/C4-architecture-model, 2018, retrieved June 14, 2024.

[12] "Haversine formula," https://en.wikipedia.org/wiki/Haversine_formula, n.d., retrieved May 25, 2024.

[13] M. T. Goodrich and R. Tamassia, *Algorithm Design and Applications*, 1st ed. Wiley, 2014. [Online]. Available: https://www.wiley.com/en-us/Algorithm+Design+and+Applications-p-9781118335918

[14] A. Horni, K. Nagel, and K. W. Axhausen, *The Multi-Agent Transport Simulation MATSim.* London: Ubiquity Press, 2016. [Online]. Available: https://doi.org/10.5334/baw

[15] M. Rieser, D. Métrailler, and J. Lieberherr, "Adding realism and efficiency to public transportation in MATSim," in *Proceedings of the 18th Swiss Transport Research Conference (STRC)*, Monte Verità, Ascona, May 2018. [Online]. Available: https://www.strc.ch/2018/Metrailler_Lieberherr.pdf

[16] OpenJDK, "Jep 454: Foreign function & memory api (second preview)," https://openjdk.org/jeps/454, 2023, retrieved September 17, 2024.

[17] "Project lombok: Cleaner, faster java," https://projectlombok.org, n.d., retrieved May 25, 2024.