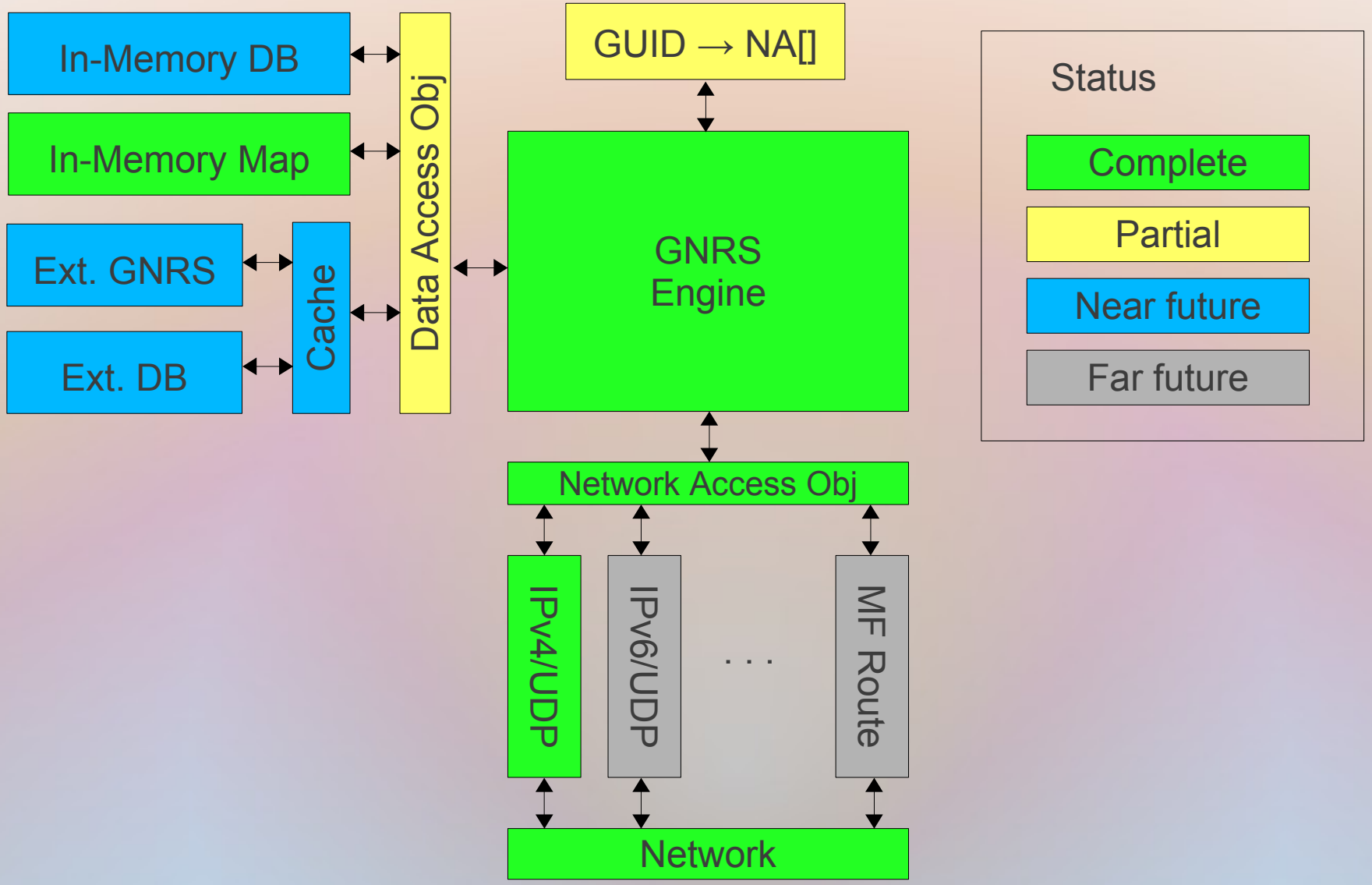
Java-Based GNRS

- Original prototype (C/C++) becoming unmaintainable, so a new implementation was chosen
- Java language selected because of familiarity, avoids some pitfalls seen in C/C++ code
- Implementation is feature-complete to about 80% of C/C++ (2 weeks)
- Uses “V1.0” protocol specification from MF Wiki
- Tracking independently in “new_server” branch of Git repository

Code Concepts

- Keep things modular - separate functionality with simple interfaces.
- Primary server engine ignorant of other component details (black box): network, GUID hashing, storage engine.
- Configuration files provide “run-time” tuning and changes wherever possible.
- Utilize standard libraries wherever practical

Server Components



Server Components

- Networking - Apache MINA (IPv4+UDP)
  - Custom JNI for MF Routing
- Longest Prefix (BGP) - Prefix Trie (3rd Party)
- Hashing - Java SE (MD5/SHA)
- Record Storage - Java SE (Map)
  - Berkeley DB/SQL

Performance Overview

- Intel i7 (jfk.rutgers.edu)
- 35-100µs processing time up to saturation
- Saturates ~70kr/s for 1 worker
- Performance is very steady
- Scales readily with 2nd worker up to NIC limit (110kr/s)

- Centaurhaul (sb1)
- 500-800µs processing time up to saturation
- Saturates ~1.1kr/s for 1 worker
- Performance highly variable
- 2nd worker causes slowdown due to contention

Performance Overview (i7 jfk)

In-memory storage

Everything resolves to local server

26K AS entries, 306K prefixes

2 Threads: Protocol, Worker

Java 1.6 (x64) 64MB Heap

Clients send 100k lookups, Stats 1/s

4,800 req/sec each client

- 4.8k/s (1): ~35µs (0% loss)
- 39k/s (8): ~57µs (1-3% loss)
- 57k/s (12): ~82µs (0-4% loss)
- 64k/s (14): ~125µs (0-3% loss)
- 69k/s (15): 130-400µs (0-2%)
- Saturates around 70k/s

2,400 req/sec each client

- 2.4k/s (1): 38µs (0% loss)
- 19k/s (8): 42µs (~0% loss)
- 38k/s (16): ~61µs (~0% loss)
- 57k/s (24): ~85µs (~0% loss)
- 70k/s (30): 150µs-2s (0.05%)
- Saturates around 70k/s

3 Threads: 2 Workers, 2 Hosts x 10/11 client = 103k/s @ 35µs (0-3%)
110k/s is limit for host (~8MB/s requests + ~4MB/s responses)

Benefits and Challenges

- Portable Code
- Large standard library
- Many 3rd-party libraries easily integrate
- Stronger OO focus makes “black boxes” easier
- Non-IP networking requires JNI
- Performance penalty ~50% compared to C/C++
- GENI image doesn’t include Java (yet?)
- Maintain separate C/C++ headers

Open Questions

- How should GUID hashing relate to networking?
  - Separate component?
  - Internal to NAO?
- How to handle GUID → GUID mapping?
  - e.g., GUID → GUID → IPv4
  - What component resolves?
- Any “session” state allowed?
- With direct responses, should “proxy” servers communicate with sender?