



# *Scaling **BIRD** Route Servers*

DENO8, Darmstadt

**Benedikt Rudolph, Sebastian Abt**

DE-CIX Research and Development

# *The BGP Internet Routing Daemon (BIRD)*

- Dynamic IP routing daemon (BGP, RIP, OSPF, Babel, BFD)
- Written in C
- OpenSource Software (GNU GPLv2)
- Widely deployed for Route Servers at IXPs
  - Mid-size to small ones
  - Few large ones with ~1000 BGP neighbors
- Flexible configuration file syntax
- Reliable and stable operation



# Motivation and Goals

Avoid:

- “**spiral of death**” syndrome
  - Re-convergence after high churn
- **performance degradation** on protocol reload / maintainance

Improve:

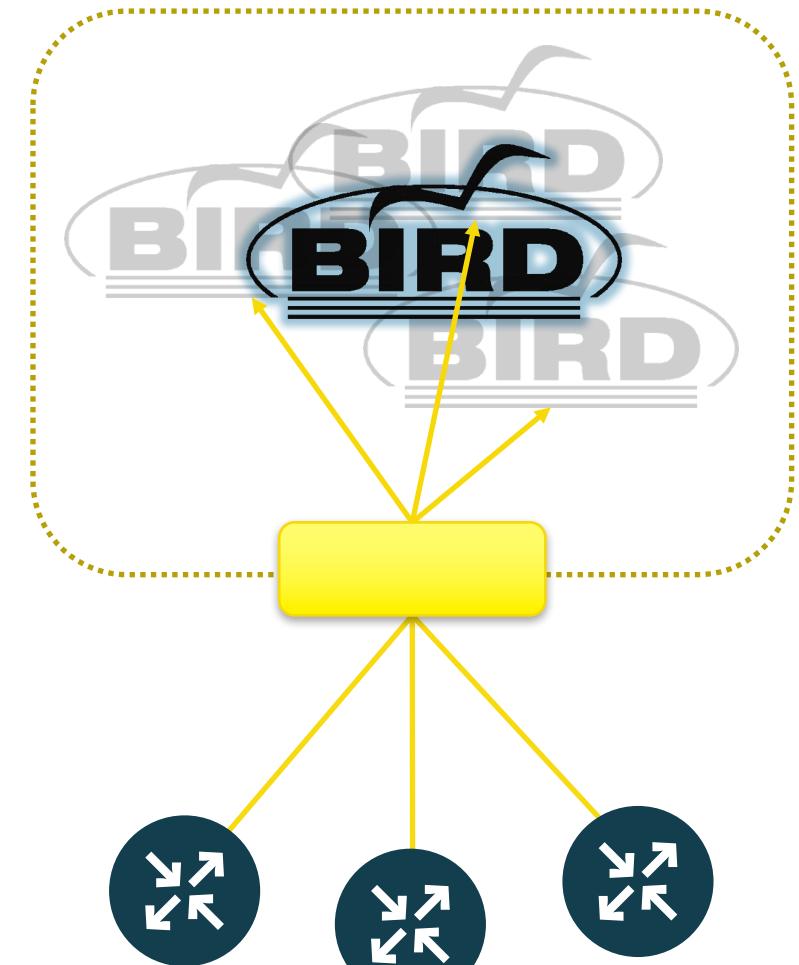
- **Scale out** to unused CPUs
- Reduce load on BIRD processes
  - Fewer peers/prefixes => faster
  - Serve more peers in total

Easy to **test, deploy and maintain**

- No client-side configuration changes
- No alterations to the source code

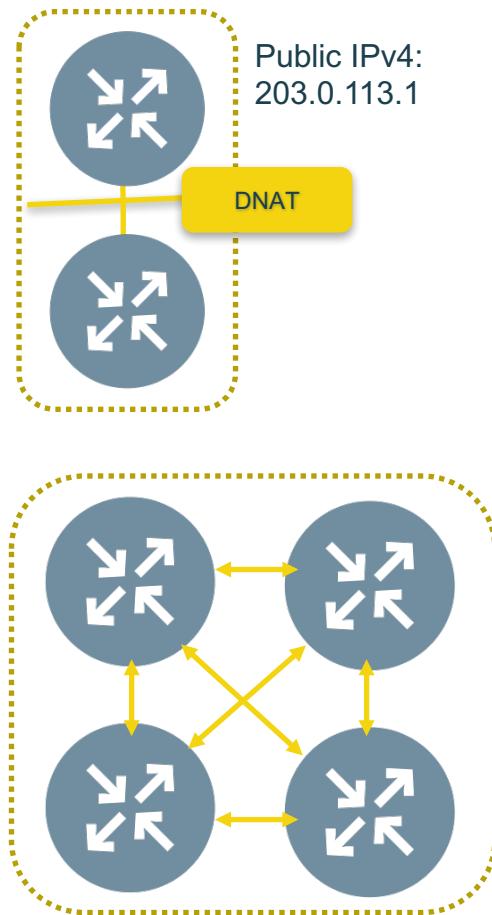
# *Problem Statement*

- Make multiple BIRD processes behave like one
  - Share a common RIB (master table)
  - Calculate the same BGP best-paths
  - Appear as the same BGP neighbor
  - Share an IP address
- Balance incoming BGP connections
  - Share load with 1, 2, 3,..., n processes



# Solution Building Blocks

- Private subnet on loopback interface
- Link master tables (full-mesh, eBGP Add-Path)
  - iBGP fails at best-path selection
  - eBGP exhibits path-hiding
- “Balance” BGP peers based on IP subnet
  - BGPv4 incompatible with TCP proxies
  - Alternative: Dynamic load balancing (iptables)
  - DestinationNAT for incoming connections

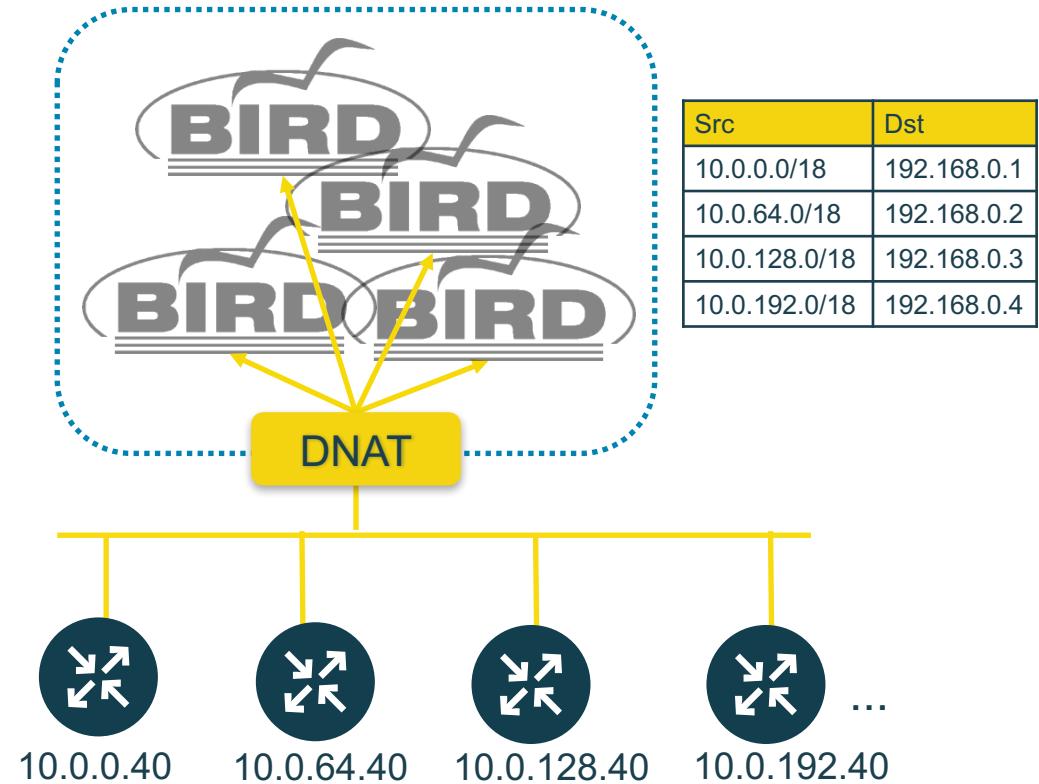


# *Details: Load Balancing*

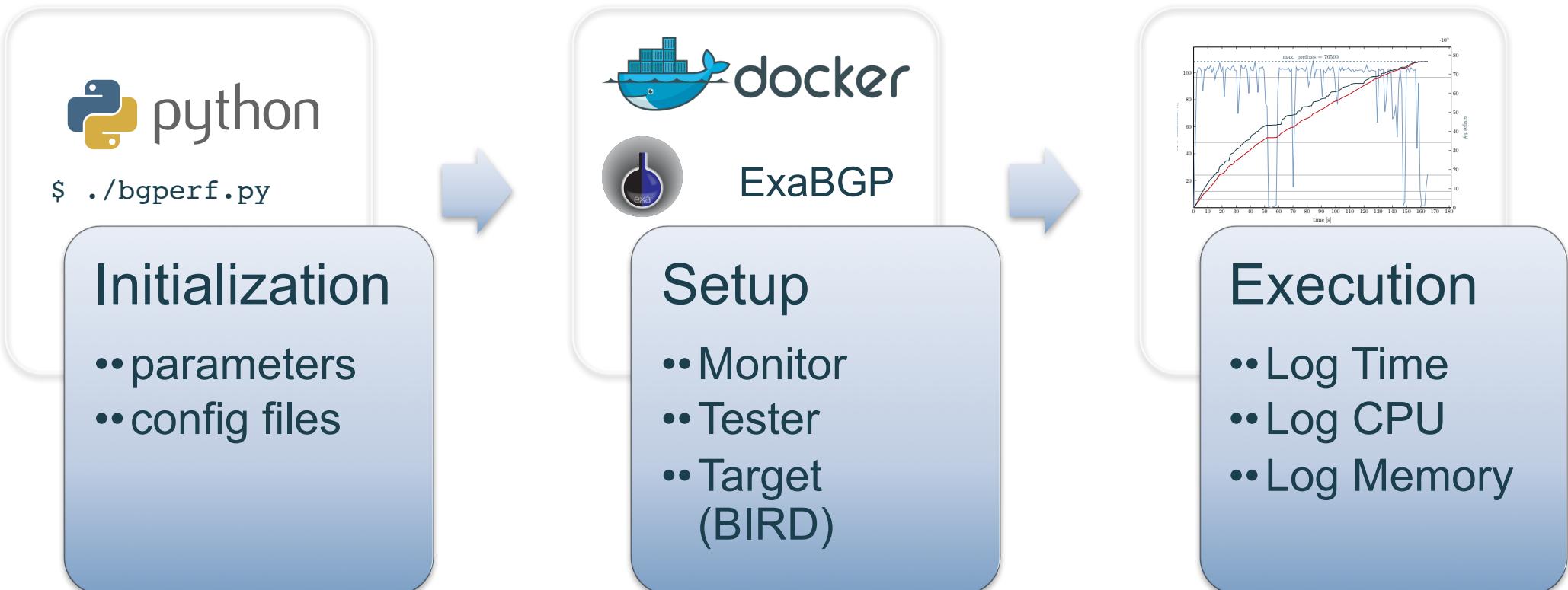
- Peering LAN split in **n** subnets
- DNAT: subnet to BIRD process

Overhead:

- Master table: **n** copies
- Best-path: **n-1** updates

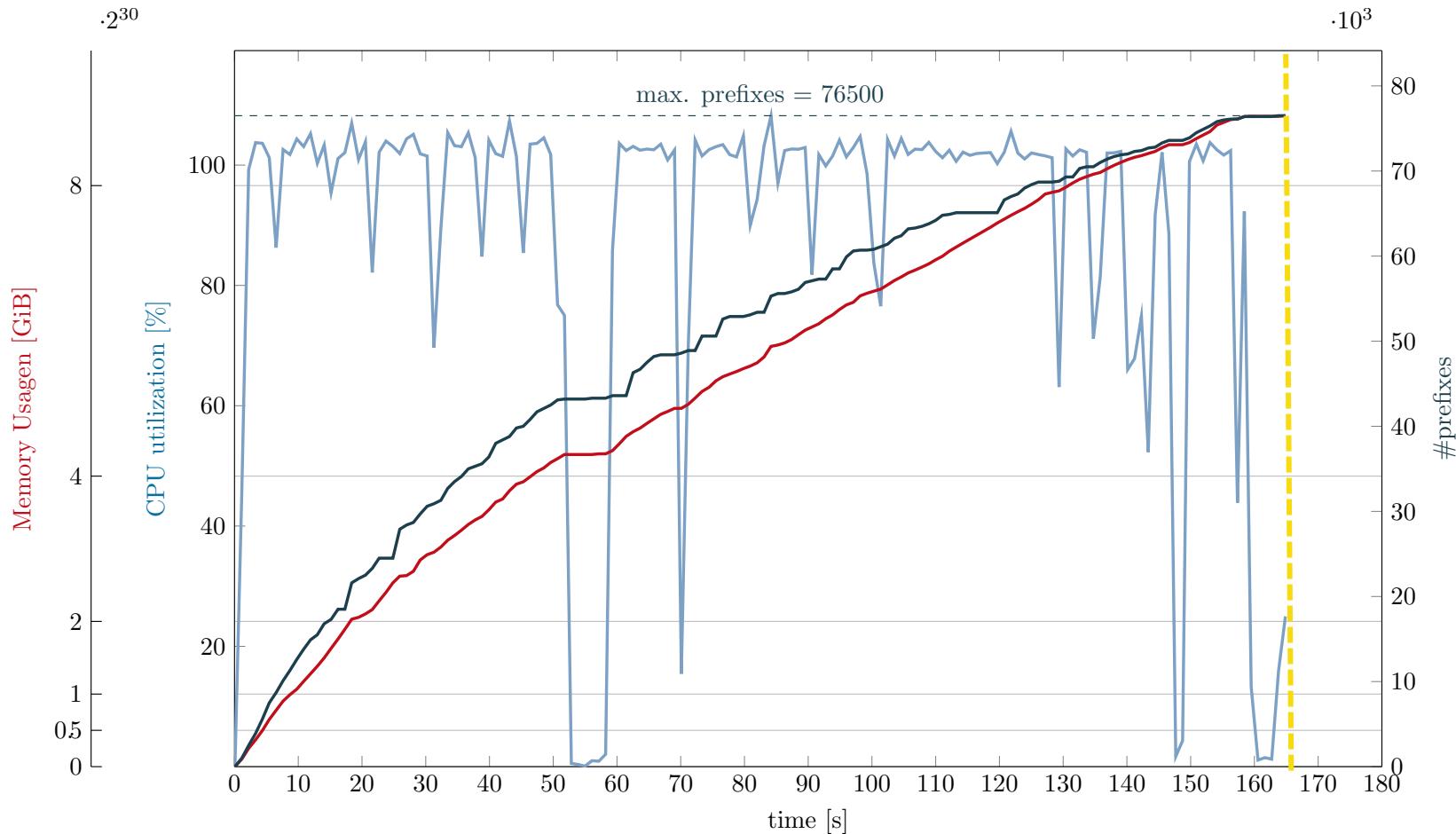


# *Test Execution with bgperf*



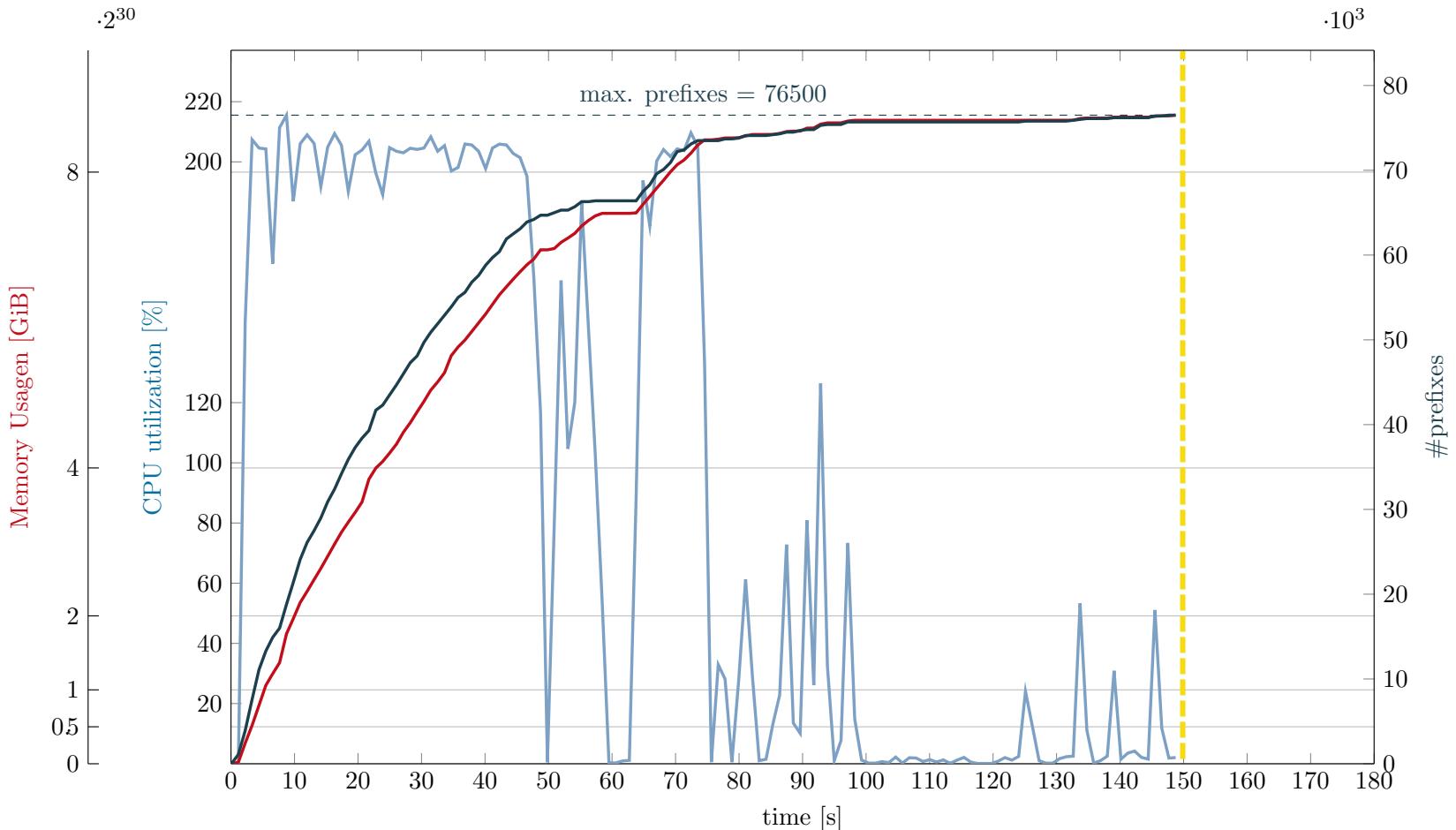
# Benchmark: Use of Multiple Processes

→ 1 BIRD Process, Time to learn 76500 prefixes = 165 s



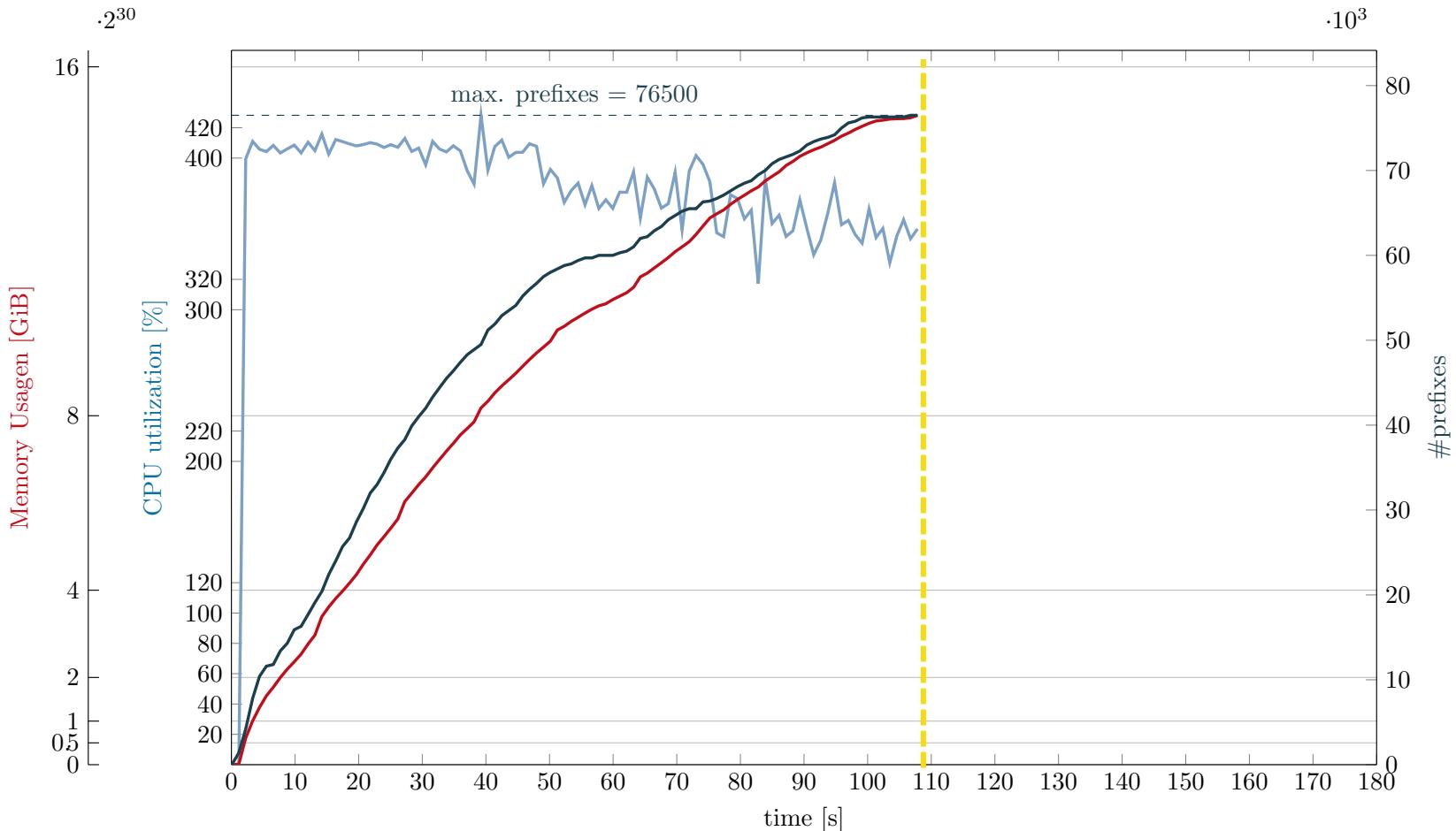
# Benchmark: Use of Multiple Processes

→ **2 BIRD Processes**, Time to learn 76500 prefixes = **150 s**



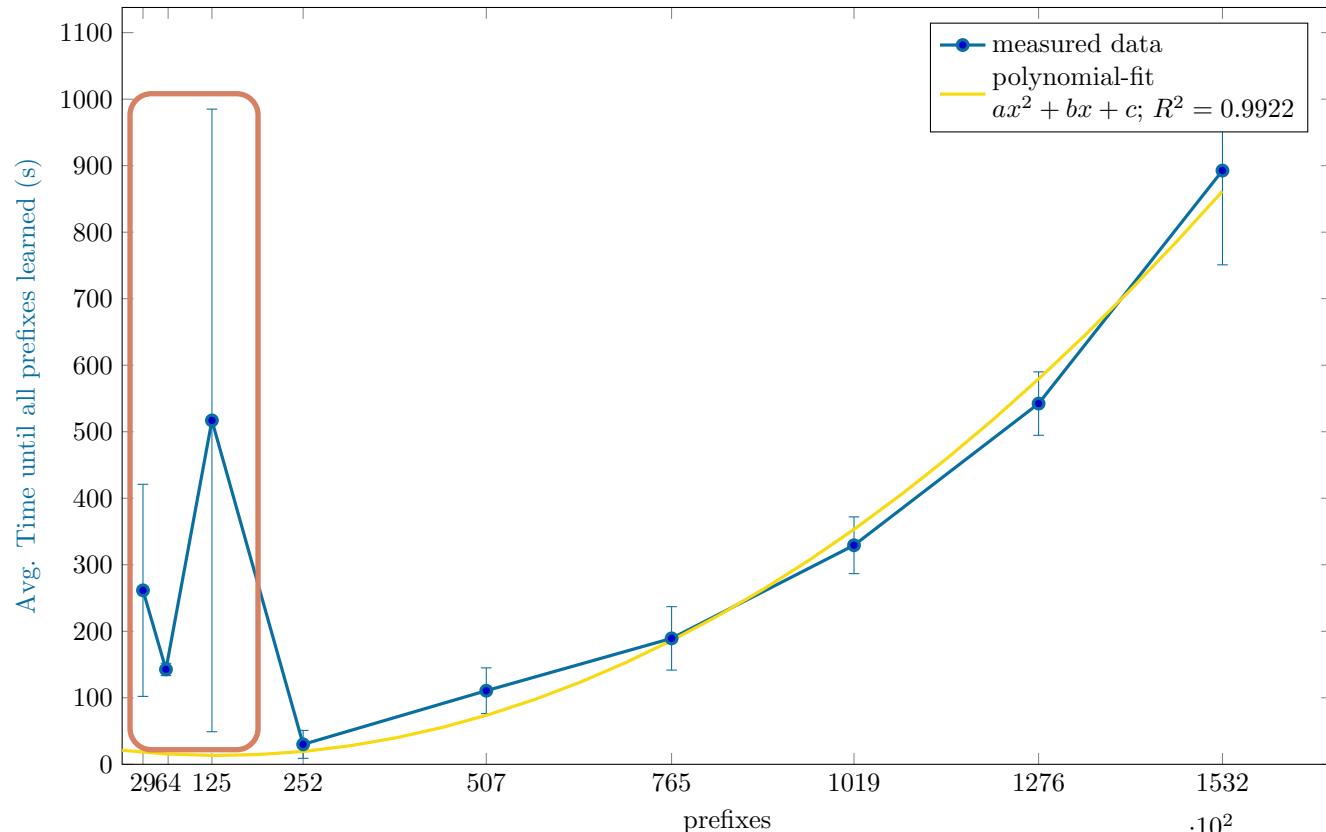
# Benchmark: Use of Multiple Processes

→ **4 BIRD Processes**, Time to learn 76500 prefixes = **108 s**



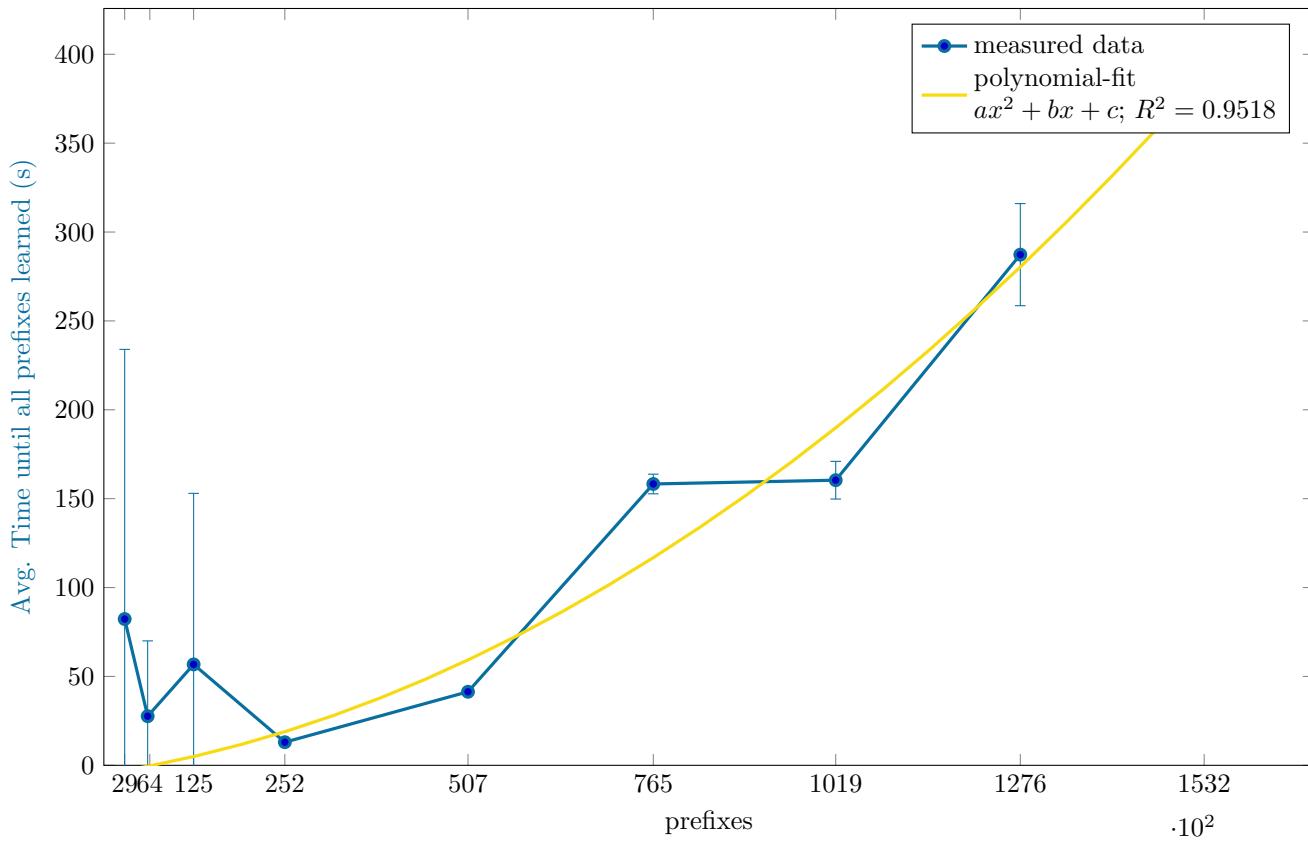
# Results: Execution Time, 1 Process

- Large variation for small settings ⇒ excluded
- Super linear scaling



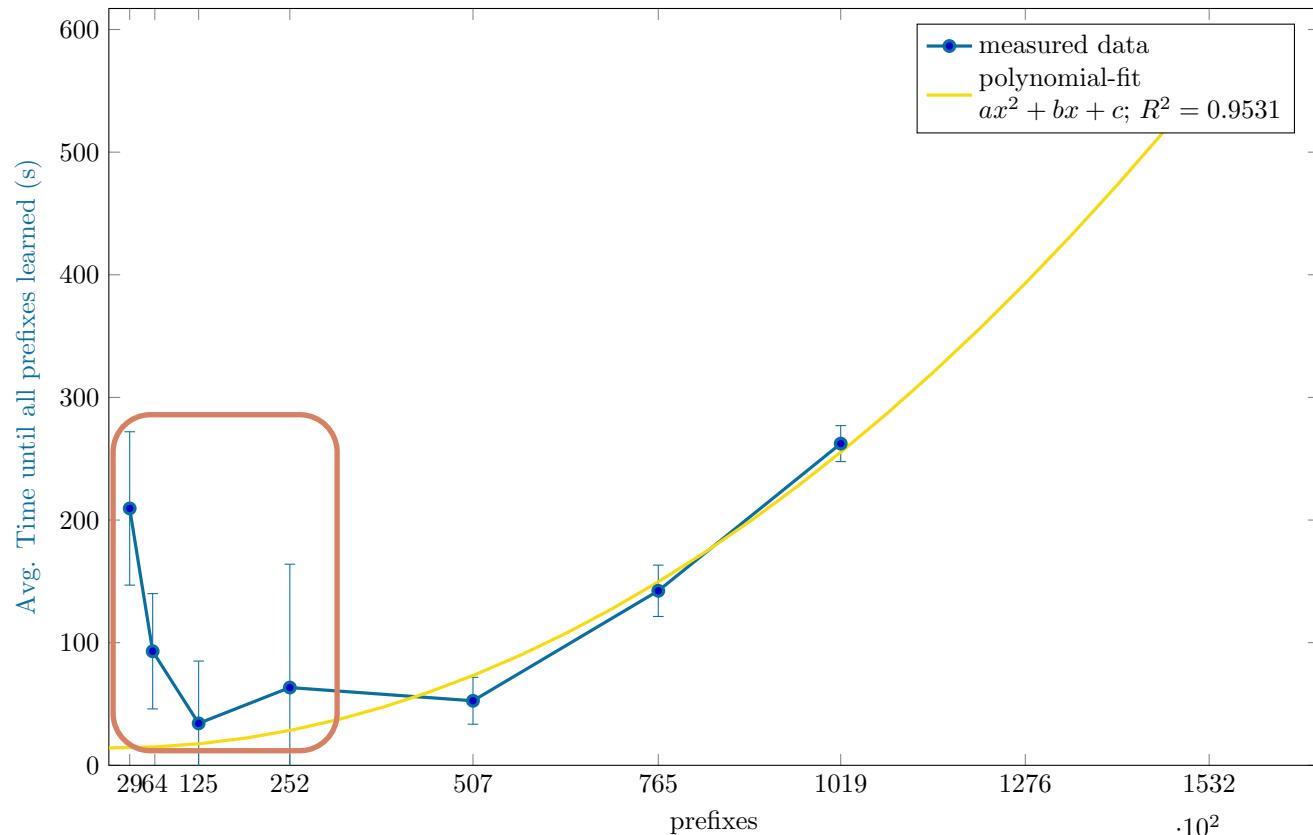
# *Results: Execution Time, 2 Processes*

- Low variation
- No results for 1532 peers ⇒ RAM limit



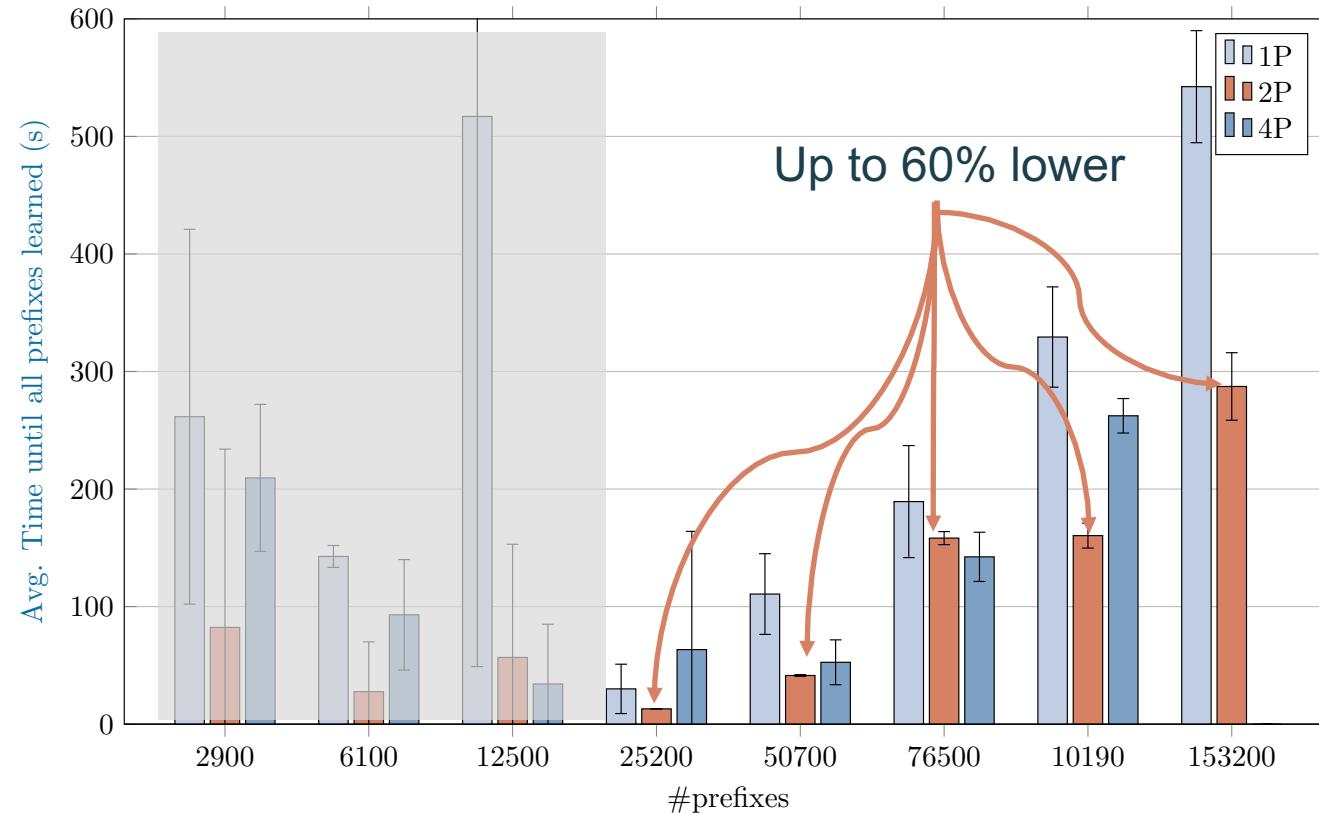
# *Results: Execution Time, 4 Processes*

- Anomaly for small settings
- No results for 1532, 1276 peers ⇒ RAM limit



# *Comparison of Execution Times*

- Anomalies for small peer count
- 2 BIRD processes: low overhead
- 4 BIRD processes: faster in some, but not all settings



## *Summary and Outlook*

- Multi-process setup improves convergence under high load
- Increased ressource usage:
  - Moderate increase in Memory (2 processes)
  - Memory usage doubles (4 processes)
- Speedup is partly egalised by overhead
  - Replication of Master Table
  - Internal Communication: TCP over loopback interface (use Sockets)
- Possible improvements / solution alternatives
  - Simple multi-threading in BIRD (e.g. per „protocol bgp“ instance)
  - Use of “realistic” peers
  - Investigate cause for “waiting“ BIRD processes (System CPU?)
  - Use a separate (physical) host for load generation (Tester)
  - Check overhead with 8 BIRD processes



# ***Questions & Answers***

## ***Talk to us!***

***rnd@de-cix.net***

**Benedikt Rudolph, Sebastian Abt**  
DE-CIX Research and Development