Der Check_MK Micro Core

22. Oktober 2014

Mathias Kettner GmbH
Linux und Open Source

Kellerstrasse 29, 81667 München • Tel.: 089 / 18 90 43 50 • info@mathias-kettner.de • www.mathias-kettner.de

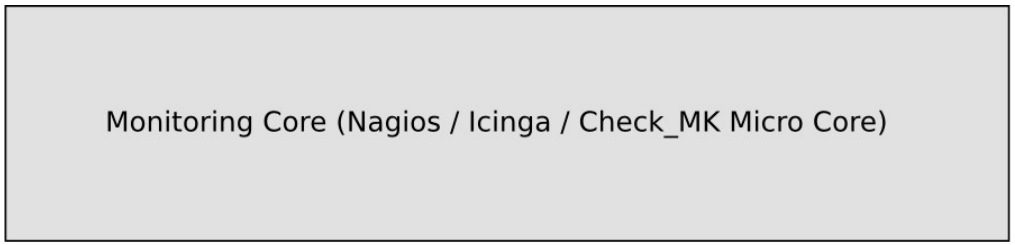
Der Check_MK Micro Core

Einfach. Schnell.

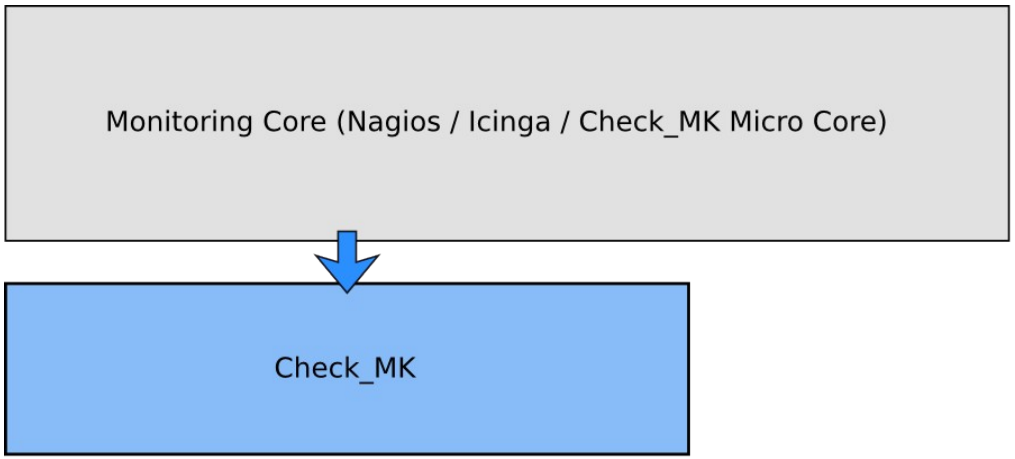
Aufgaben des Cores

- Anstoßen von Check-Plugins
- Empfangen von Resultaten
- Speichern der Resultate
- Erkennen von Änderungen (OK -> CRIT)
- Auslösen von Aktionen (Notifikationen)
- Weiterleiten von Performancedaten an RRDs
- Aktuelle Daten für Abfragen bereitstellen

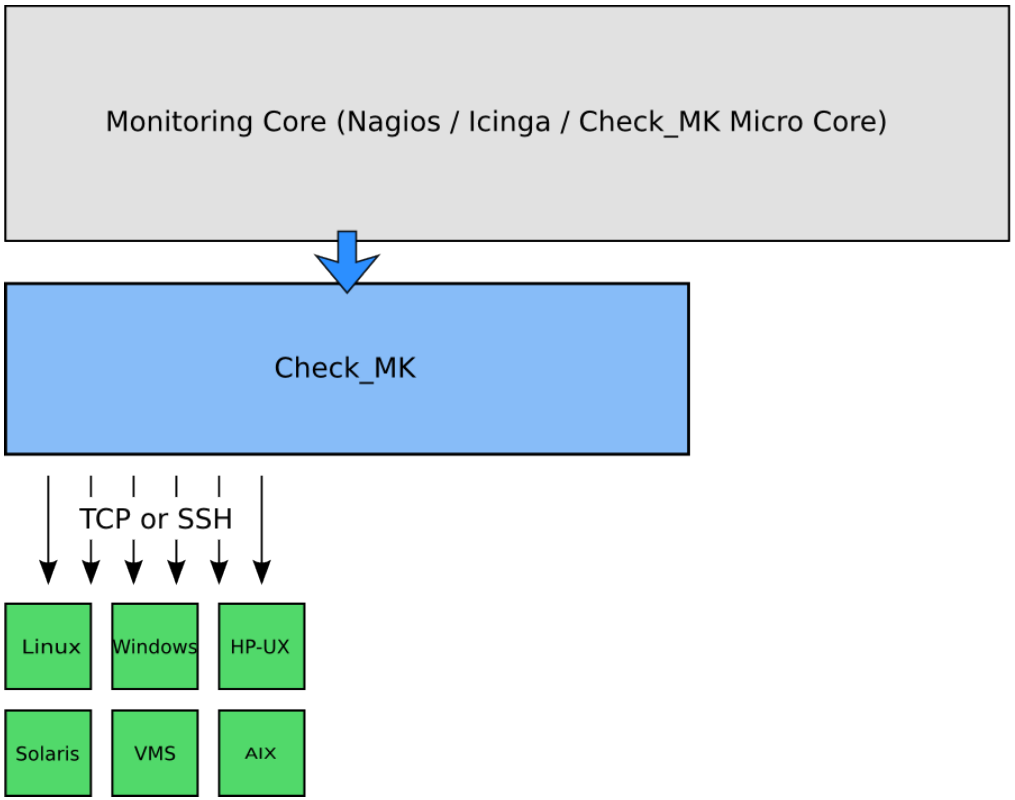
Stellung des Cores im System



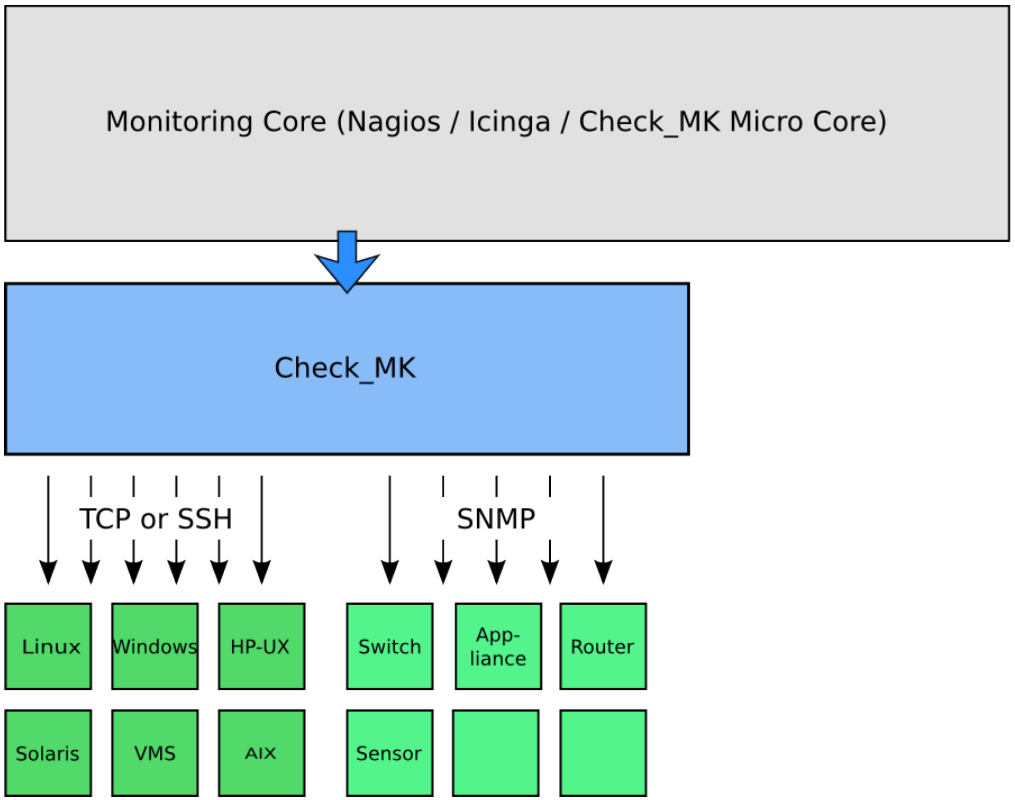
Stellung des Cores im System



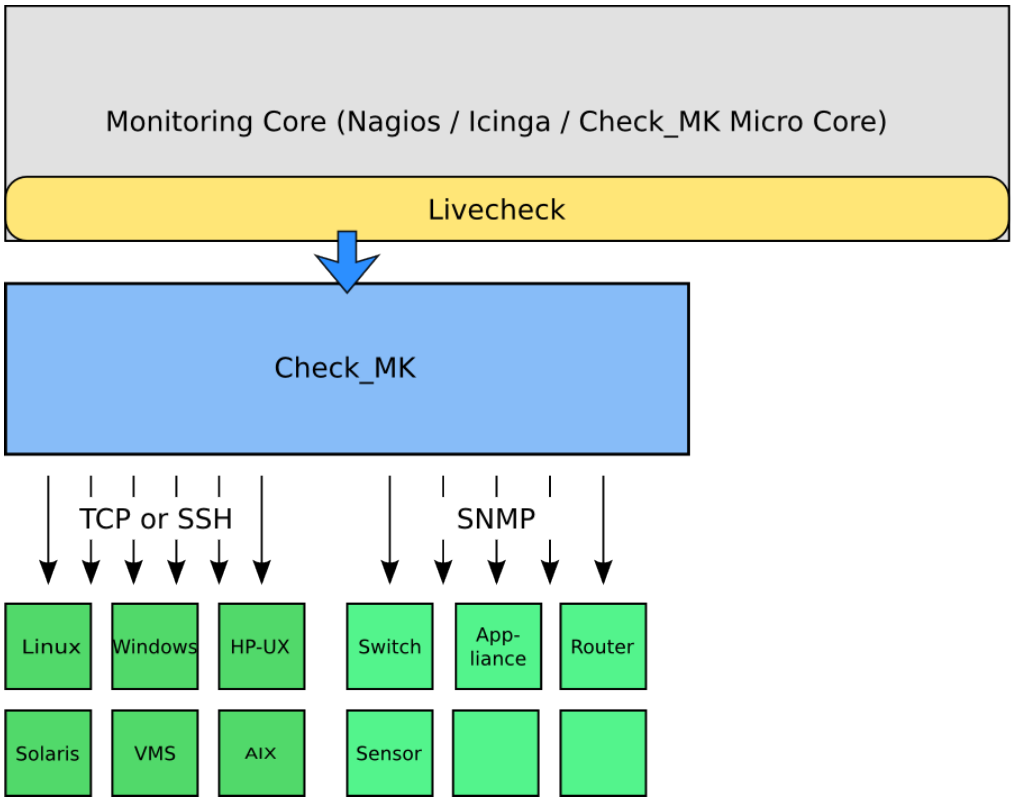
Stellung des Cores im System



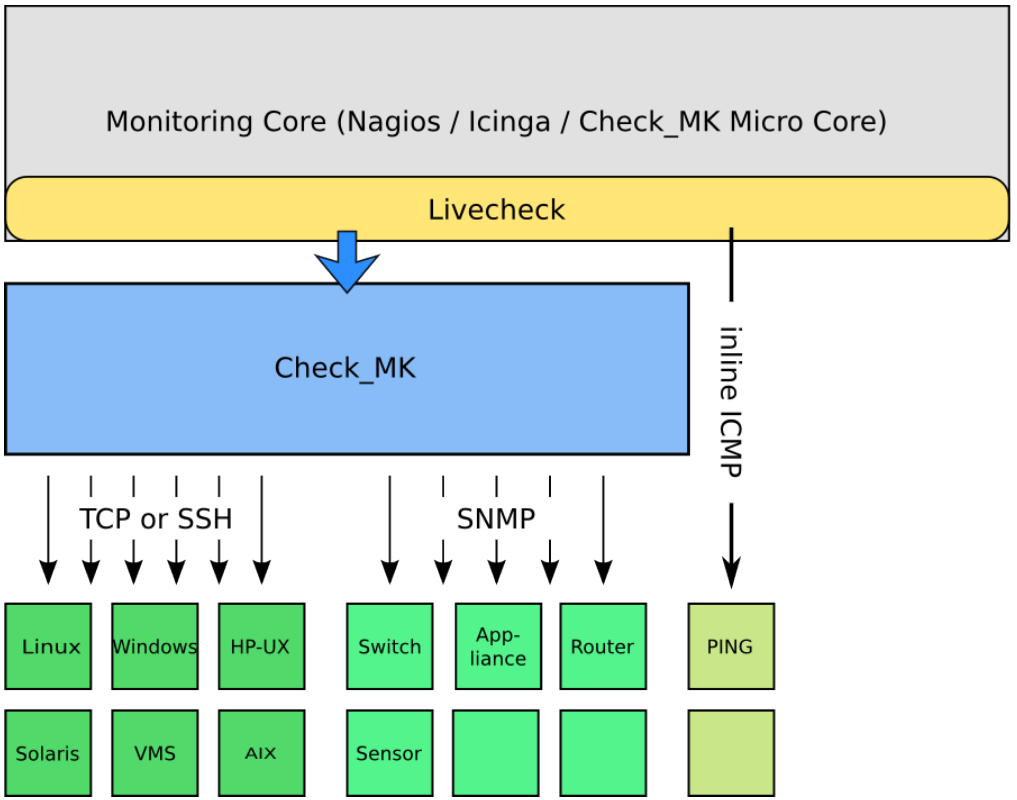
Stellung des Cores im System



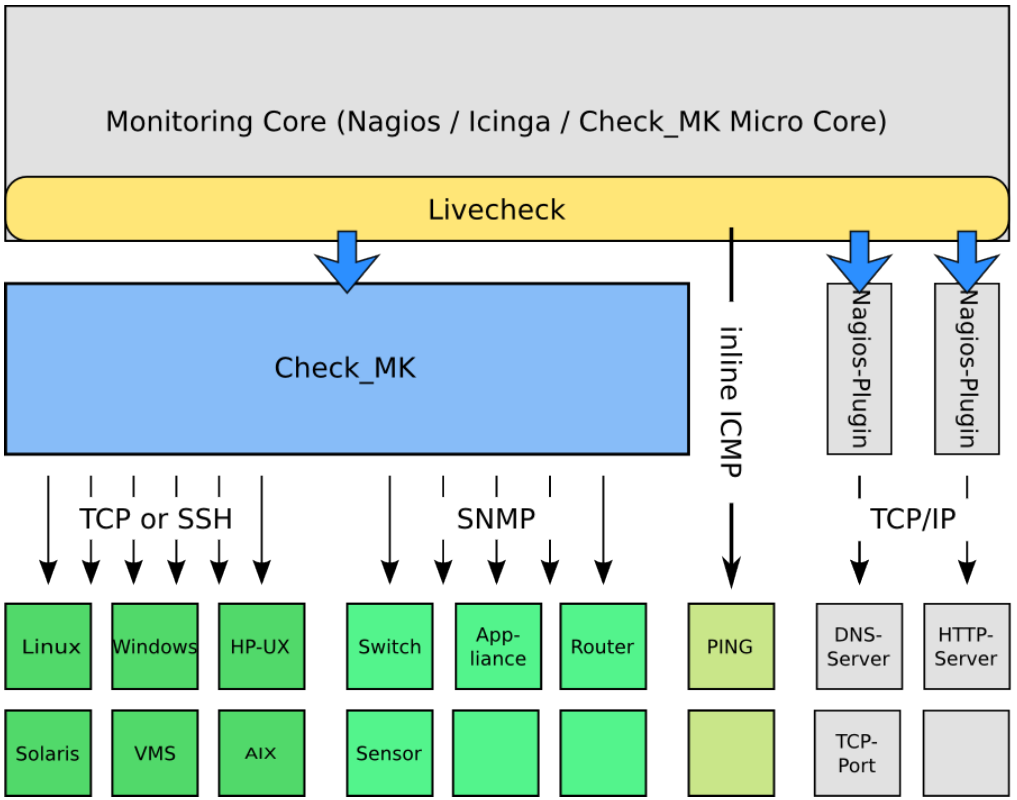
Stellung des Cores im System



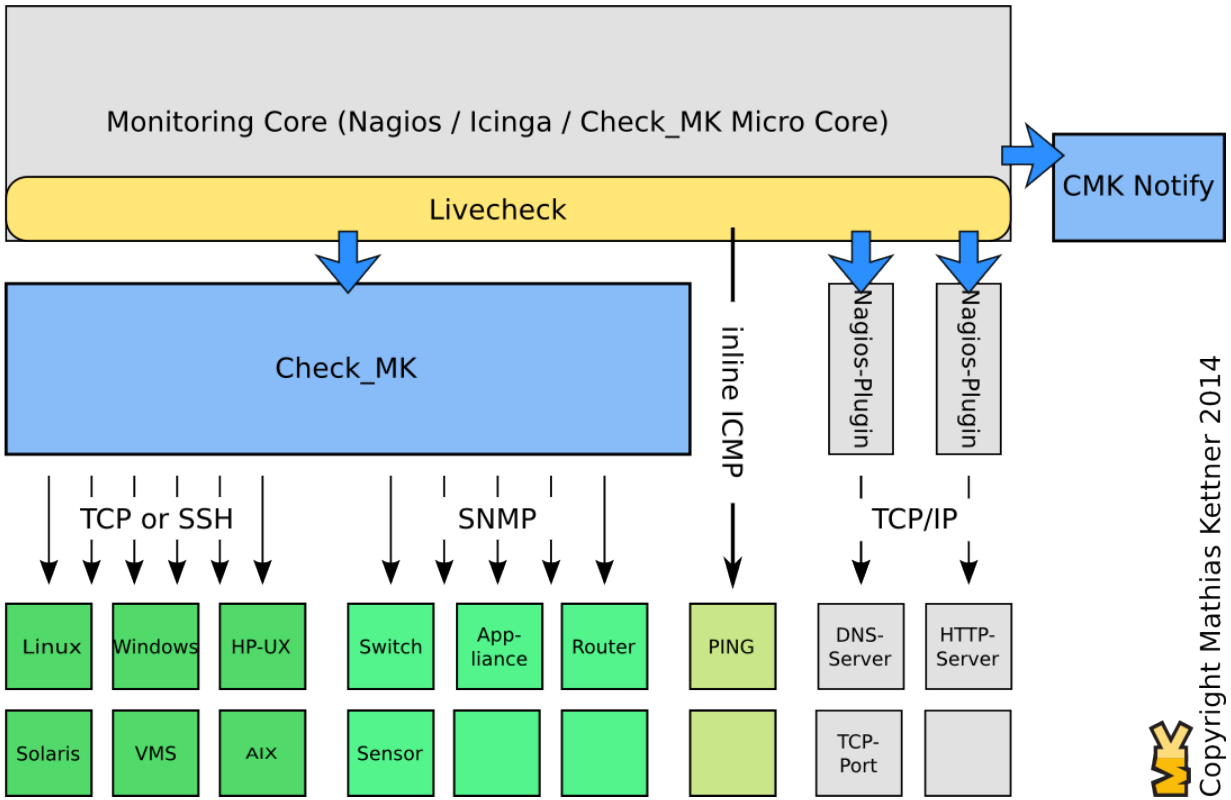
Stellung des Cores im System



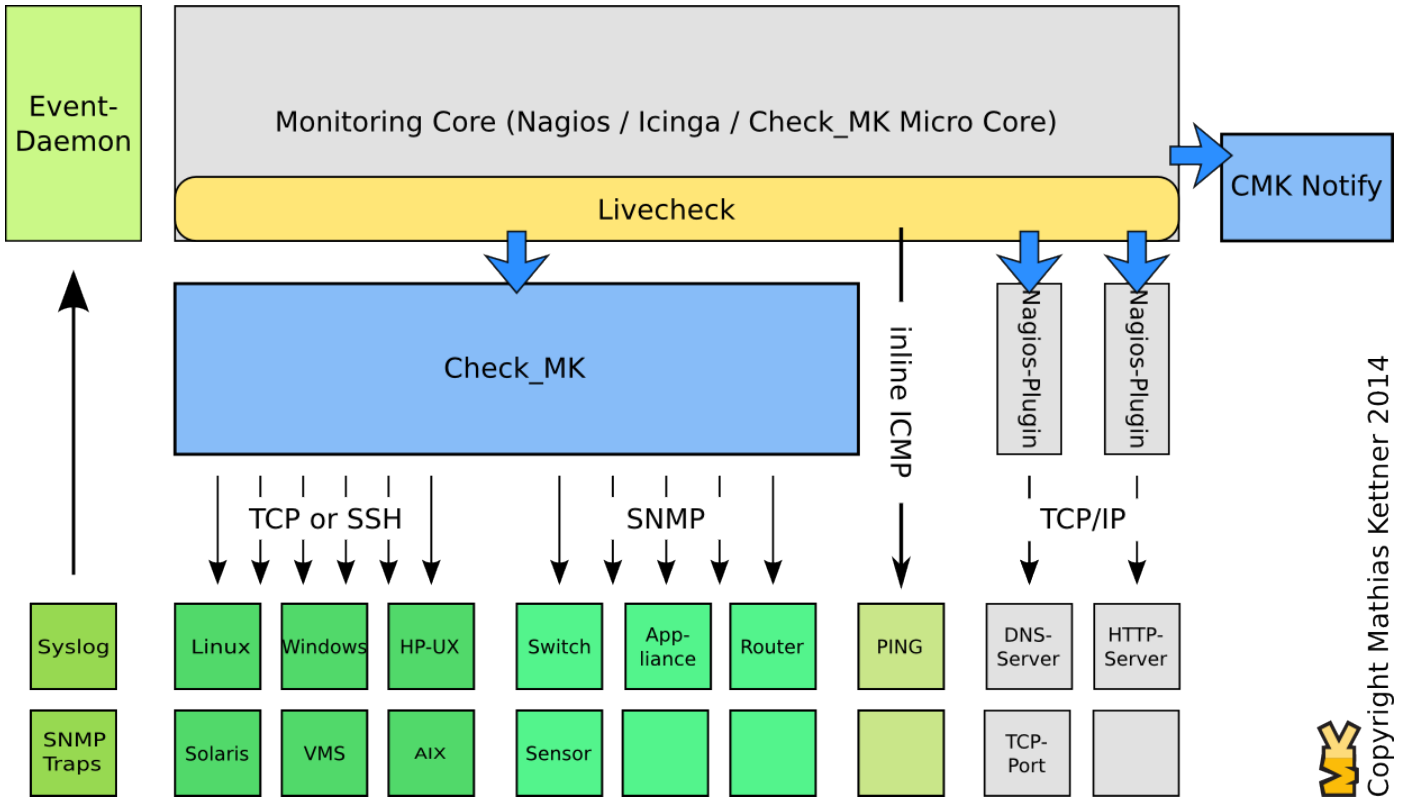
Stellung des Cores im System



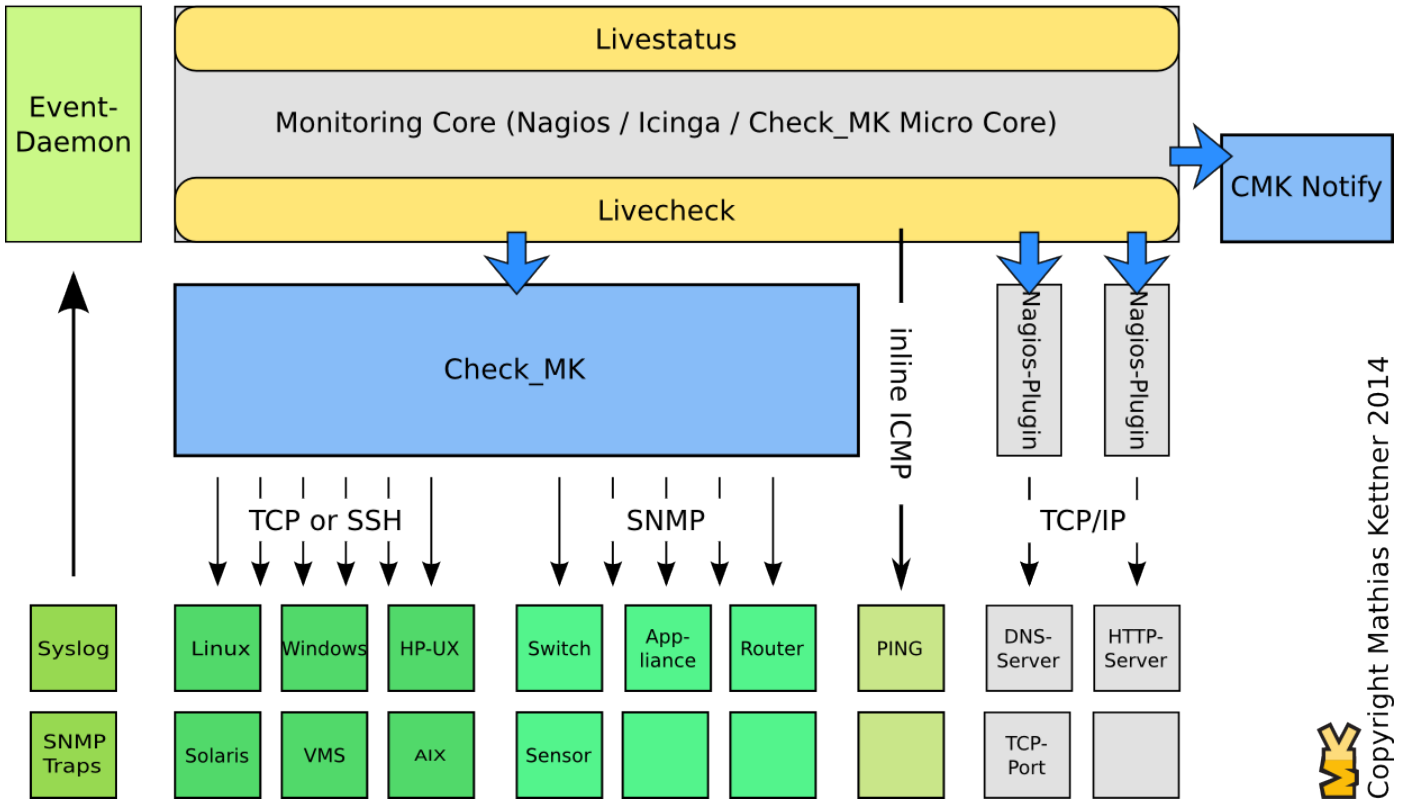
Stellung des Cores im System



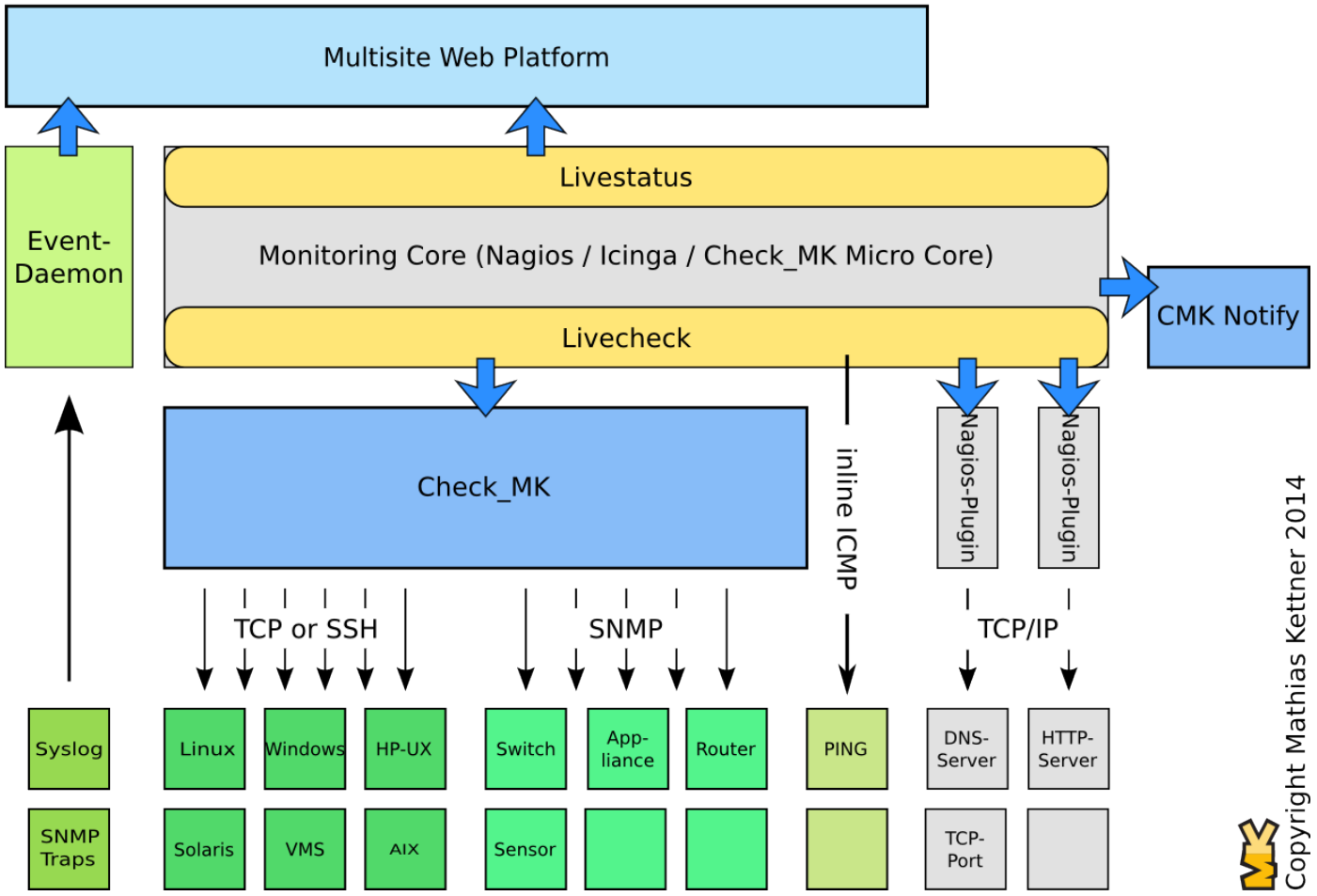
Stellung des Cores im System



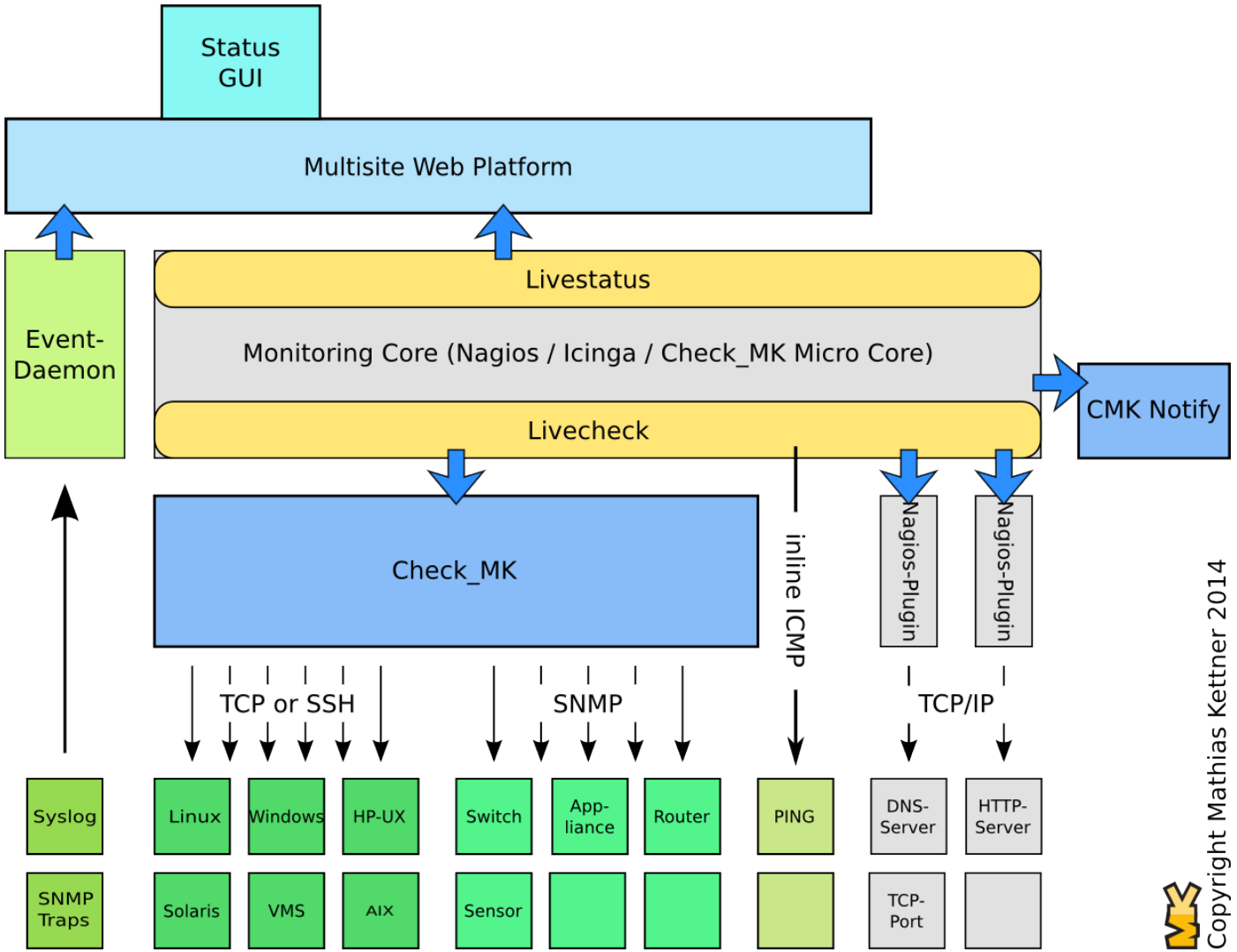
Stellung des Cores im System



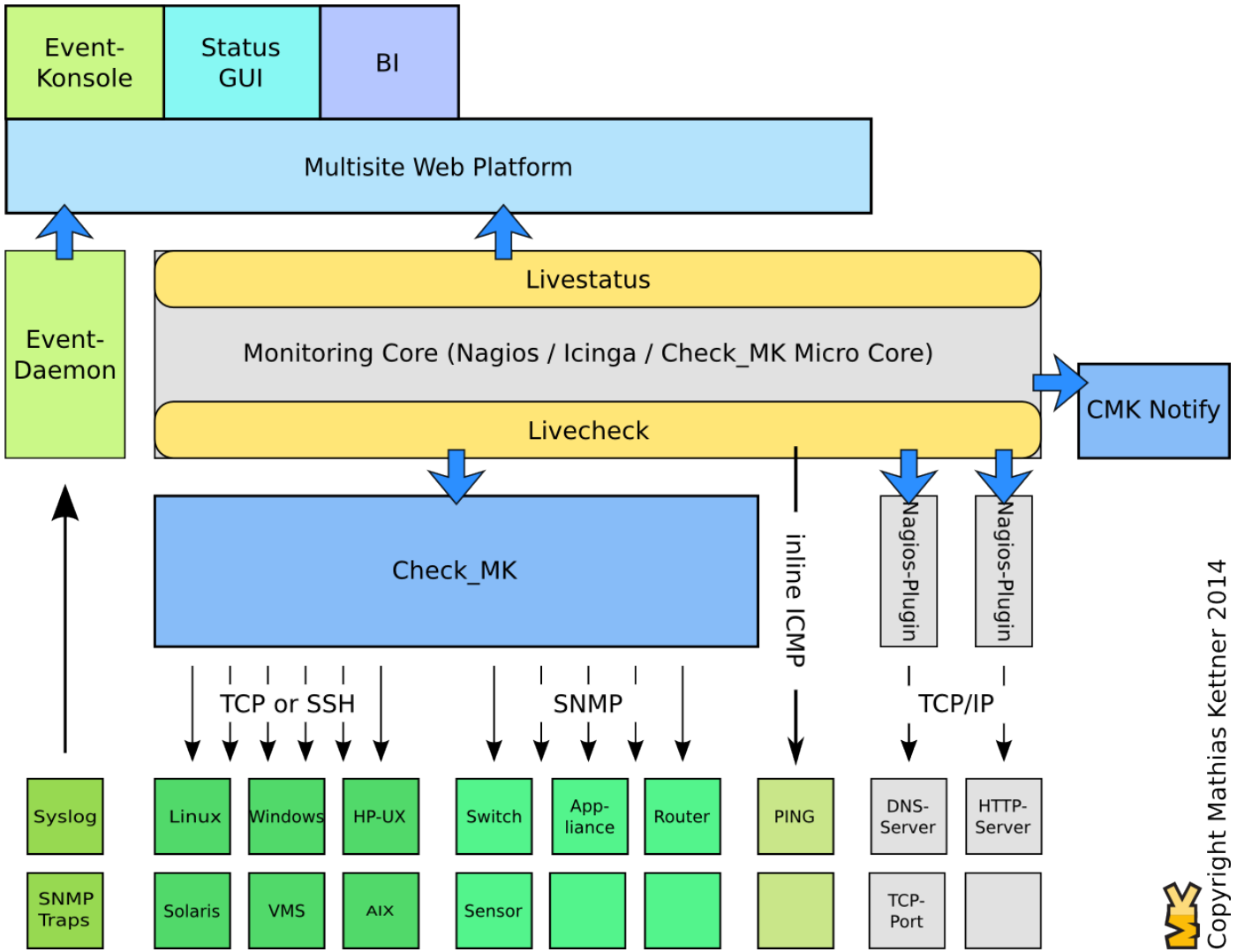
Stellung des Cores im System



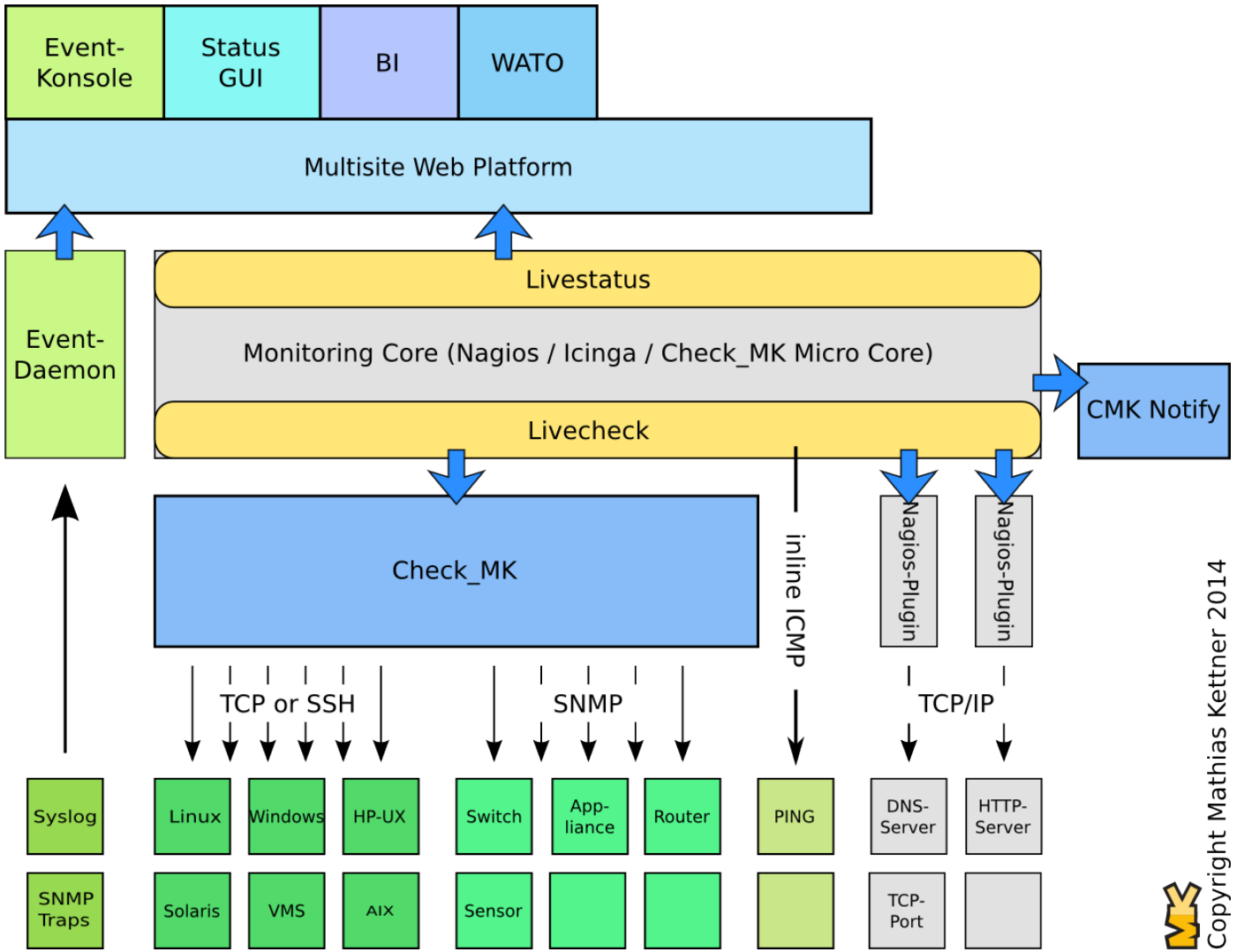
Stellung des Cores im System



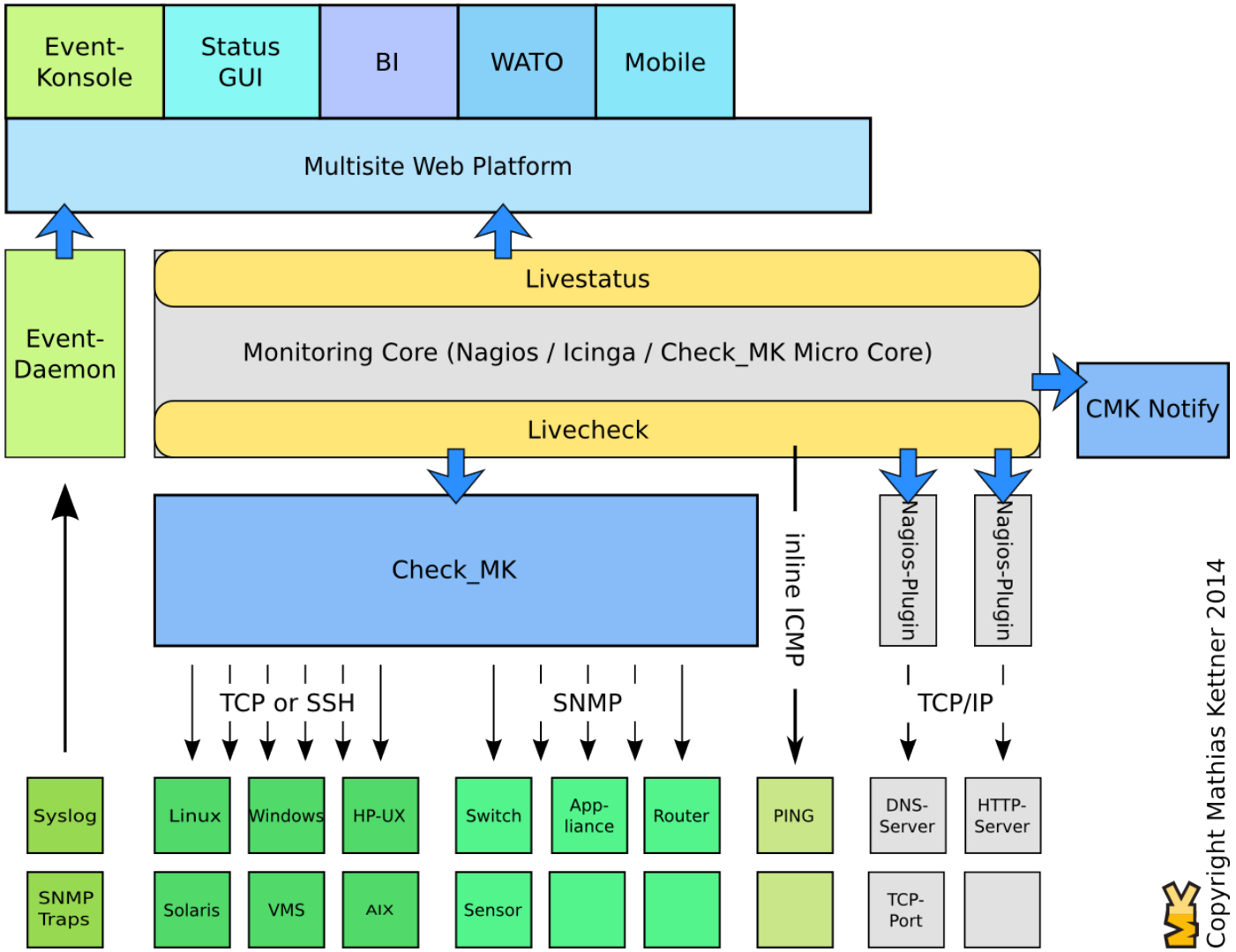
Stellung des Cores im System



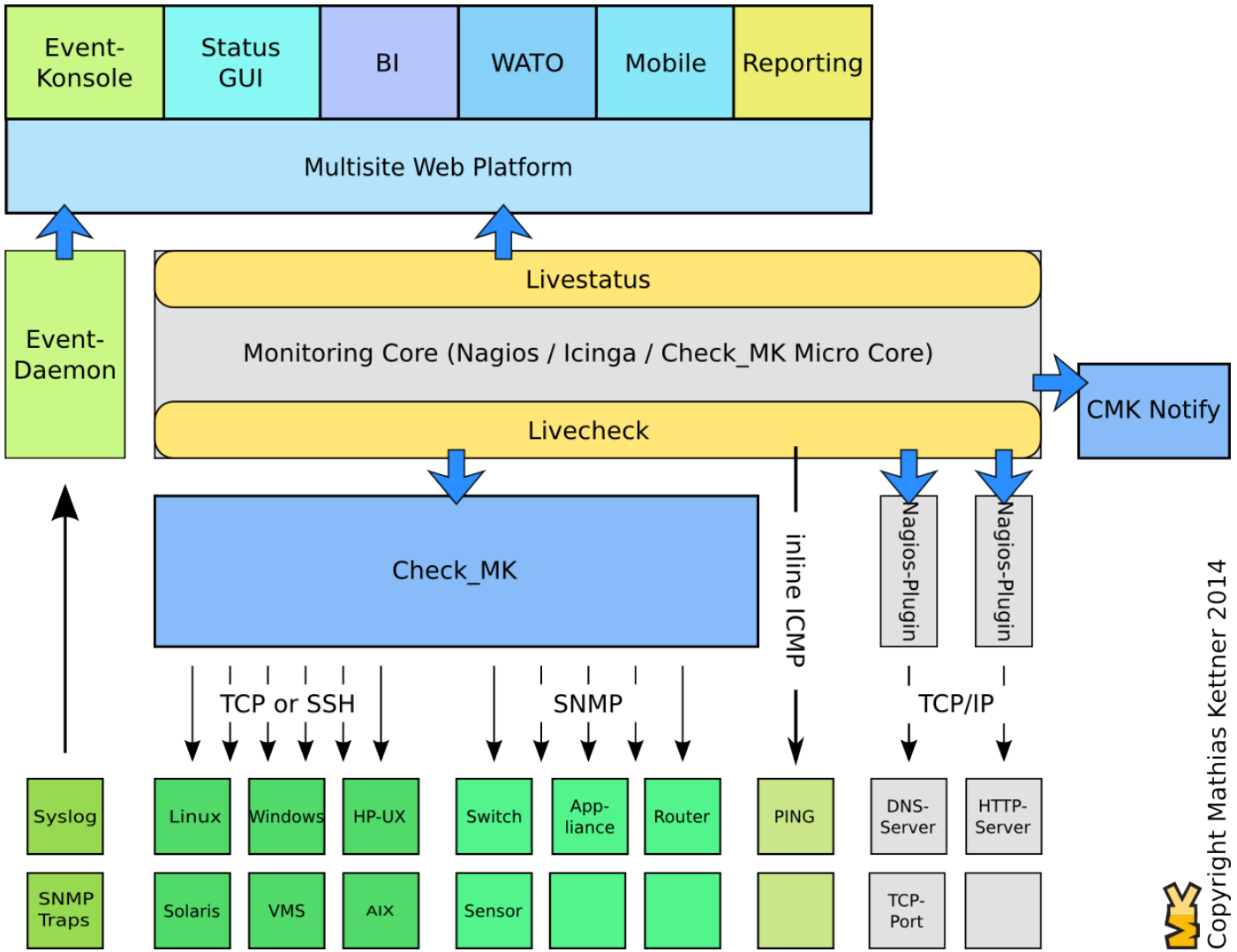
Stellung des Cores im System



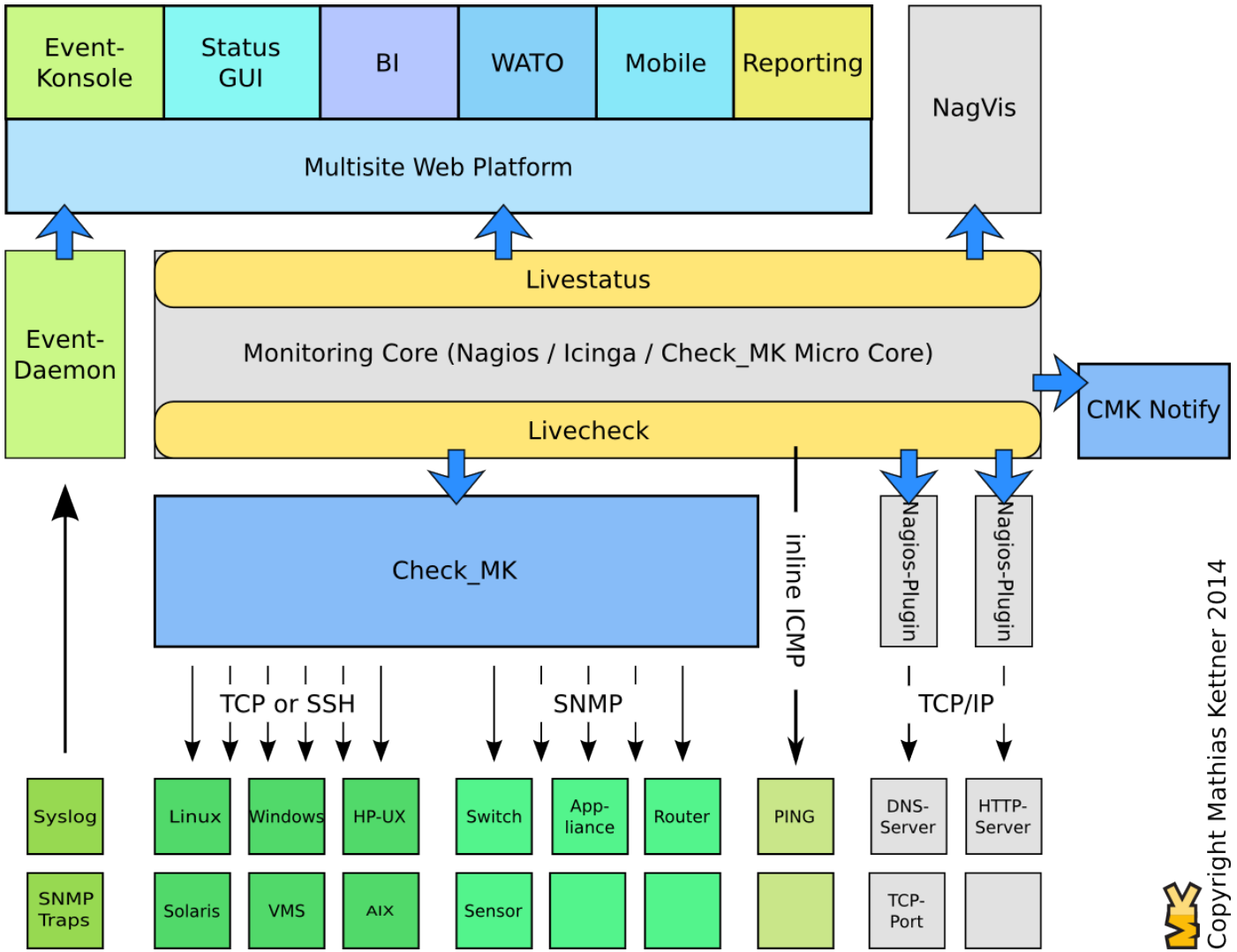
Stellung des Cores im System



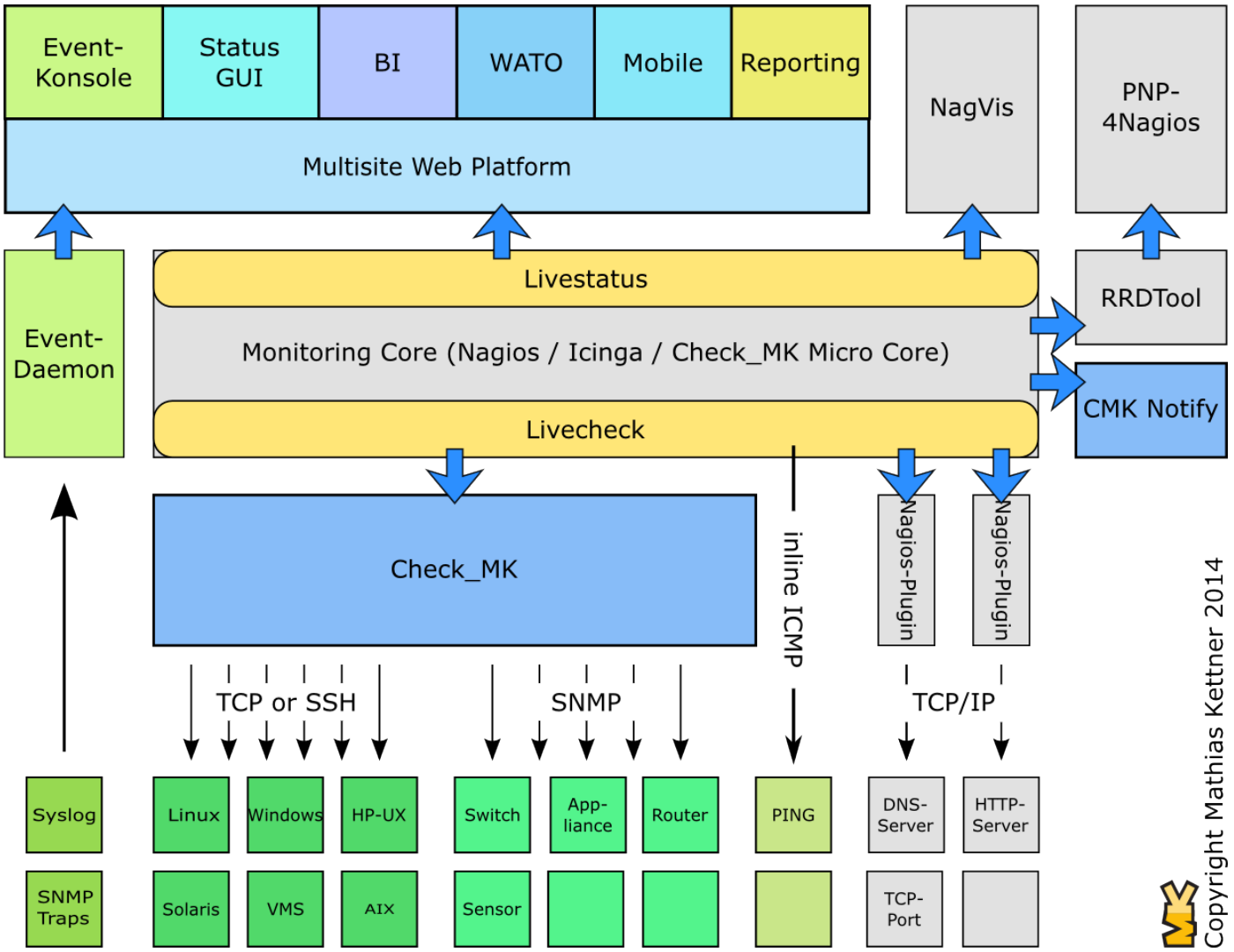
Stellung des Cores im System



Stellung des Cores im System



Stellung des Cores im System



Nagios und Derivate

- Am Anfang: Nagios Core
- Erster Fork: Icinga
- Neuentwicklung in Python: Shinken
- Zweiter Fork: Naemon
- neugeschrieben in C++: Icinga 2
- *Reicht's nicht langsam?*

Warum nicht Nagios?

Nichts gegen die Leistung von Ethan, aber:

- Code von Nagios und Forks sehr umfangreich
- großflächige Copy & Paste-Programmierung
- Programmierung mit vielen Seiteneffekten
- Code ist nicht Thread-sicher
- Code ist hässlich und kaum wartbar
- Konfigurationsänderung erfordert Neustart

Warum nicht Shinken?

- Python leichter wartbar, aber..
- C/C++ ist view schneller (richtig programmiert)
- Architektur ist recht komplex
- Und: Abhängigkeit von fremdem Projekt

Designziele vom CMC

1. Einfach
2. Schnell

1. Einfach

- Komplexität aus dem Code auslagern
- Kein Konfigurationsparser
- Timeperiods offline berechnen
- Keine Modulschnittstelle
- Livestatus fest eingebaut

Ergebnis:

Nur 8% der Codezeilen von Nagios

2. Schnell

- Monitoring ohne Prozesserzeugung!
- Konfiguration und Status in Binärform
- Intern gehen alle Zugriffe über Indizes
- Optimale Unterstützung von Check_MK
- Keine Temp-Dateien oder Spoolfiles

2. Schnell

Ergebnis:

drastische Reduktion der CPU-Auslastung

Config von 600.000 Services in 0,5 sec laden

10.000 Checks / Sekunde und mehr

Einschränkungen

- aktuell keine Eventhandler
- keine Service Dependencies
- kein Lesen von Nagios-Konfiguration
- keine Brokermodule

Sehr wohl geht

- Ausführen von „normalen“ Nagios-Checks
- Eskalationen (über Check_MK)
- Weiterarbeiten mit bestehenden RRDs
- Alles, was Check_MK so braucht

Wer alles über WATO konfiguriert, hat keine Enschränkungen.

Dinge, die nur CMC kann

Ein paar Dinge kann nur der CMC, z.B.

- Performancedaten an Graphite senden
- Konfiguration der RRD-Struktur pro Host/Service
- Availability Cache
- Konfiguration neu laden ohne Stoppen des Cores oder von laufenden Checks

Migration auf CMC

Was muss man tun, um von Nagios auf CMC zu wechseln?

Event Handler

Automatisches Ausführen von Aktionen wird aktuell nur durch Notifications unterstützt

- Bei Bedarf also Notification-Skripte erstellen

Service Dependencies

- Mal ehrlich, wer braucht die?
- Umständlich zu konfigurieren
- Sehr intransparent
- Check_MK geht eh einen anderen Weg
- Beispiel: neuer ORACLE Agent

Legacy Checks

- Klassiche Nagios-Checks gehen wunderbar, aber...
- legacy_checks += versteht CMC nicht
- Grund: diese basieren auf define command {}
- Also: Umstellen auf custom_checks
  - Das ist das Format, das auch WATO nutzt
- Umstellung ist auch mit Nagios kompatibel und schadet nicht

Vorher: legacy_checks

main.mk (altes Format)

```
# Definition of the Nagios command
extra_nagios_conf += r"""

define command {
    command_name    check-my-foo
    command_line    $USER1$/check_foo -H $HOSTADDRESS$ -w $ARG1$ -c $ARG2$
}
"""

# Create service definition
legacy_checks += [
  (( "check-my-foo!20!40", "FOO", True), [ "foohost", "othertag" ], ALL_HOSTS ),
]
```

Nachher: custom_checks

main.mk (neues Format)

```
custom_checks += [
  ({
      'command_name':         'check-my-foo',
      'service_description': 'FOO',
      'command_line':         'check_foo -H $HOSTADDRESS$ -w 20 -c 40',
      'has_perfdata':         True,
  },
  [ "foohost", "othertag" ],
  ALL_HOSTS )]
```

Eskalationen

Werden von CMC nicht unterstützt, aber...

...durch Check_MK extern gelöst:

- Flexible Notifications (alt)
- Rule Based Notifications (neu)

Weitere Migration

Weiterhin zu beachten bei der Migration:

- Historie durch Kopieren von Dateien übernehmen
- Aktueller Zustand von Downtimes, Acknowledgements wird übernommen
- Aktueller Status wird **nicht** übernommen (aber ist dann in 1 Minute wieder da)
- Weitere Einzelheiten: http://mathias-kettner.de/cms_cmc_migration.html

Host Checks - Smart PING

Der CMC bietet einen neue Methode für Host-Checks:

Smart PING

Smart PING

Ideen von Smart PING:

- **Ein** Prozess wickelt alles Pingen ab
- Keine Prozesserzeugung mehr
- PINGs gleichmäßig verteilen, nicht in Bursts
- Auch eingehende TCP SYN/RST/ACK auswerten!

Smart PING - Vorteile

Vorteile:

- Praktisch keine CPU Last mehr
- Zeitnahe Hostchecks
- Kein nachgelagerter Hostcheck bei TCP-“Connection refused“ nötig

Smart PING - Nachteile

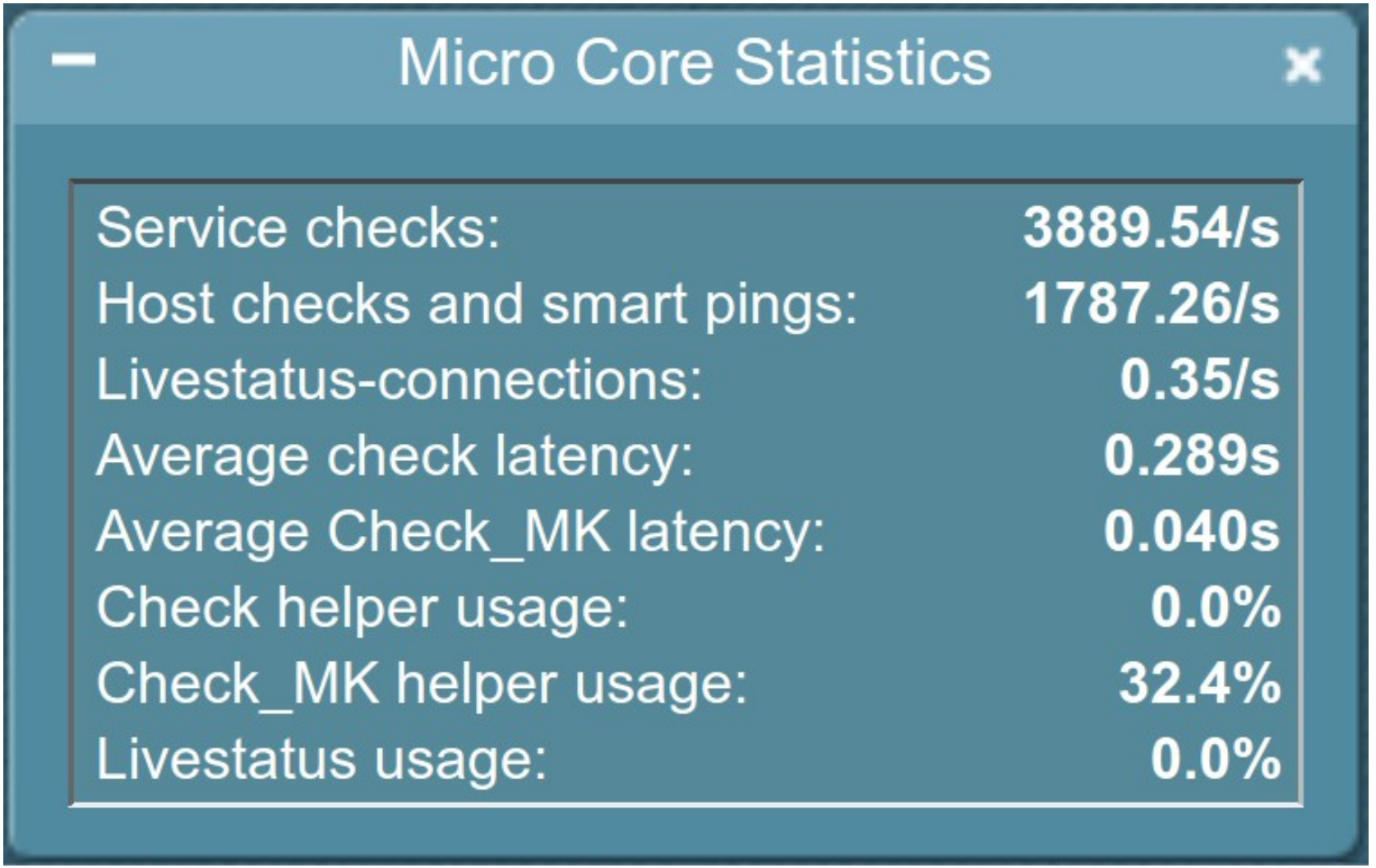
Nachteile:

- Smart PING sammelt keine Performancedaten!

Macht, aber nix, weil:

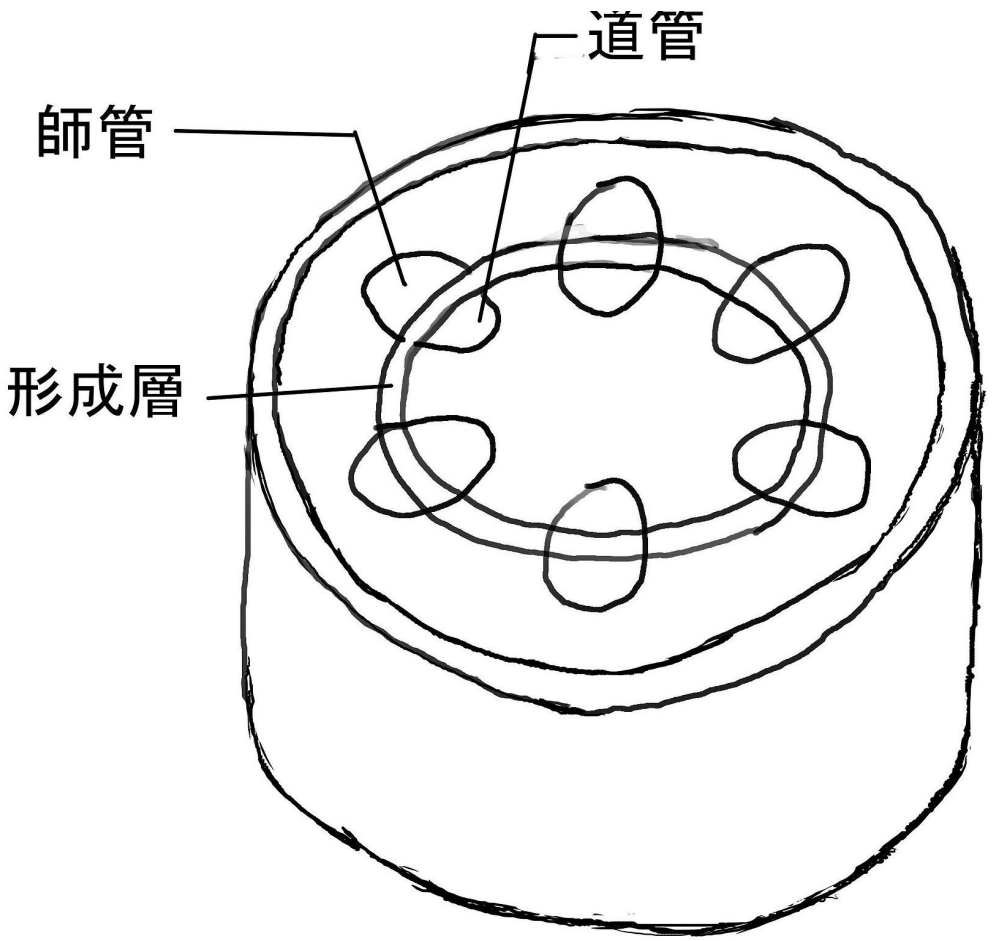
- Bei benötigten Hosts, entweder PING-Service
- oder Hostcheck auf check_icmp zurückstellen

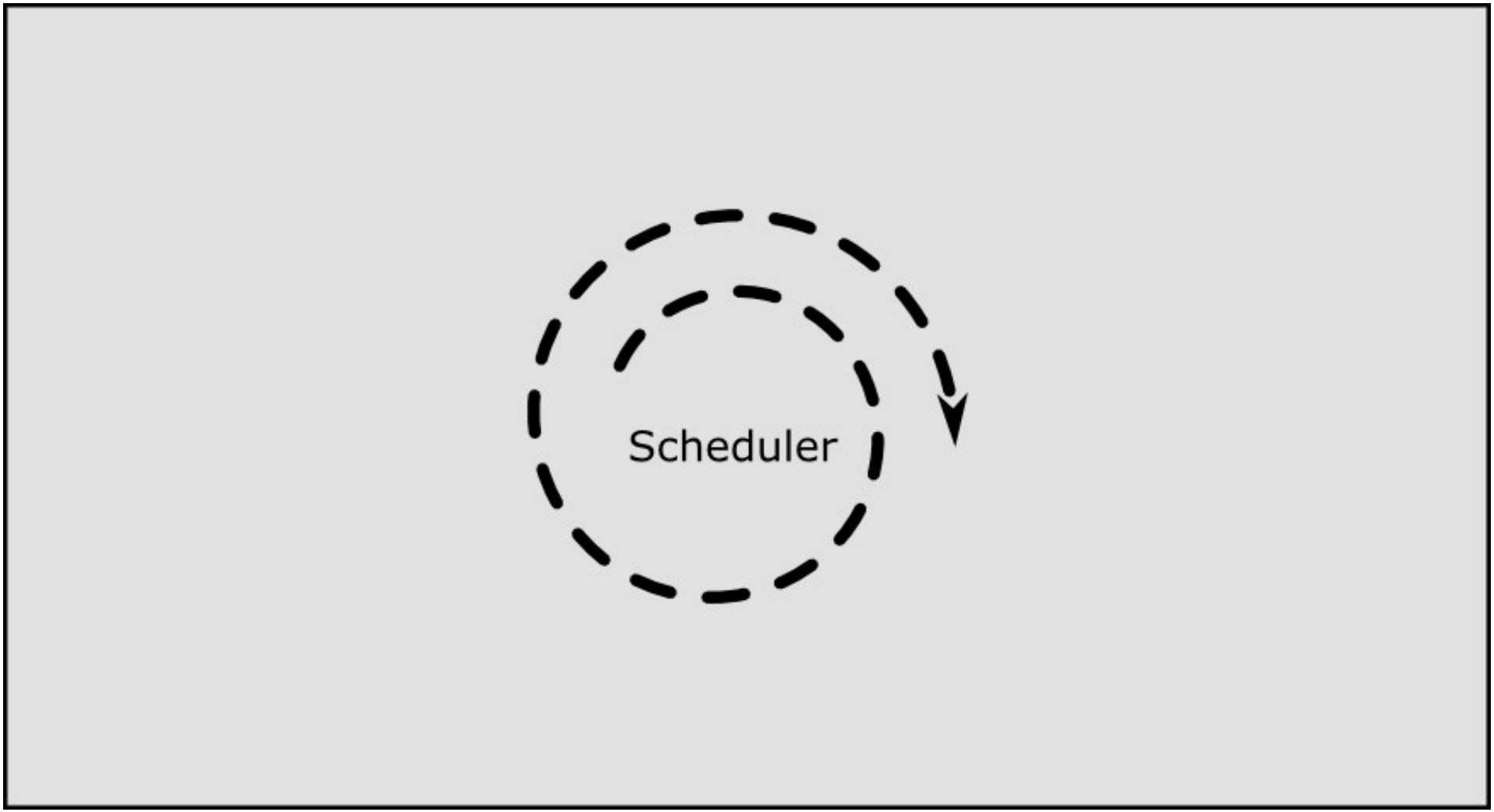
Snapin zur Core-Statistik



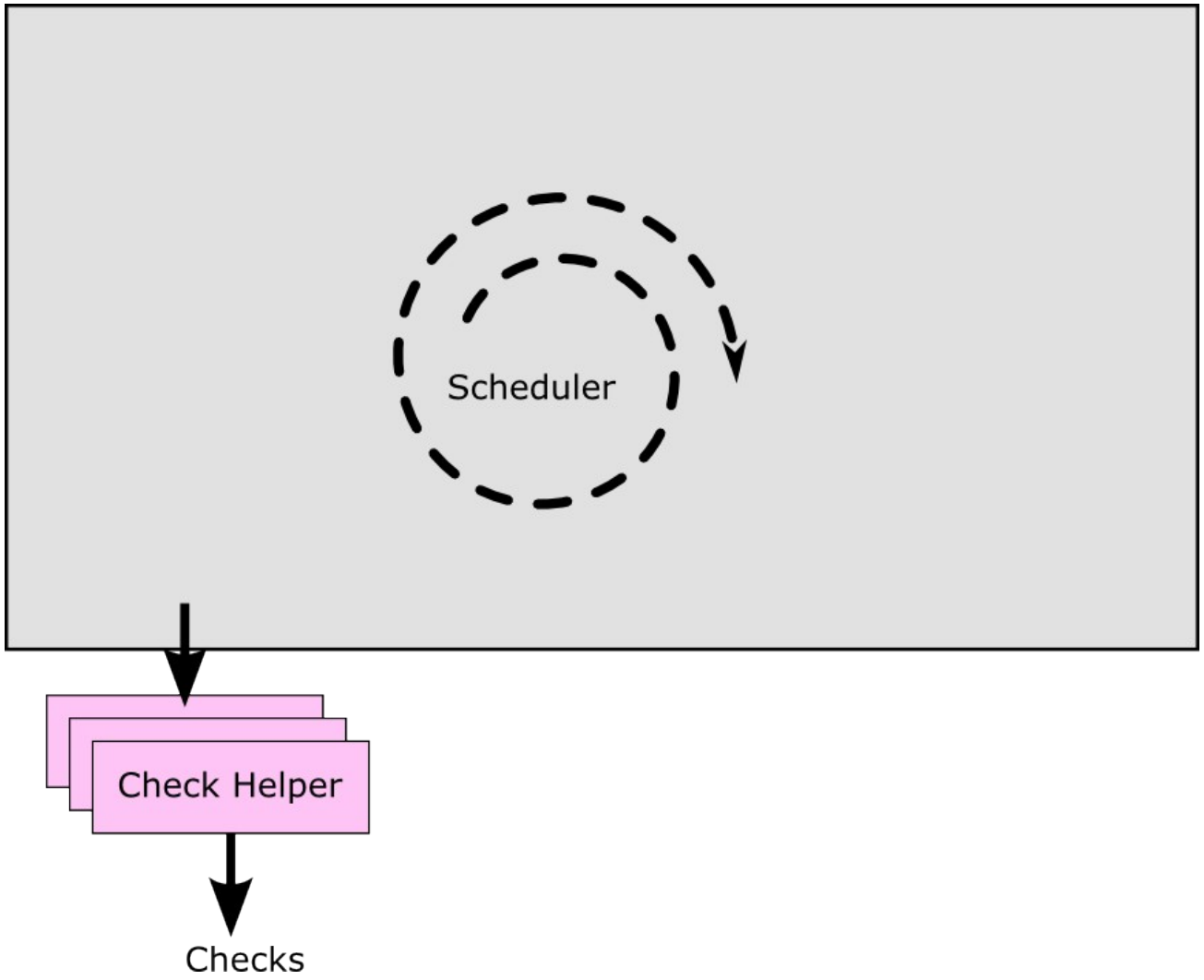
Die Architektur des CMC

CMC Architektur

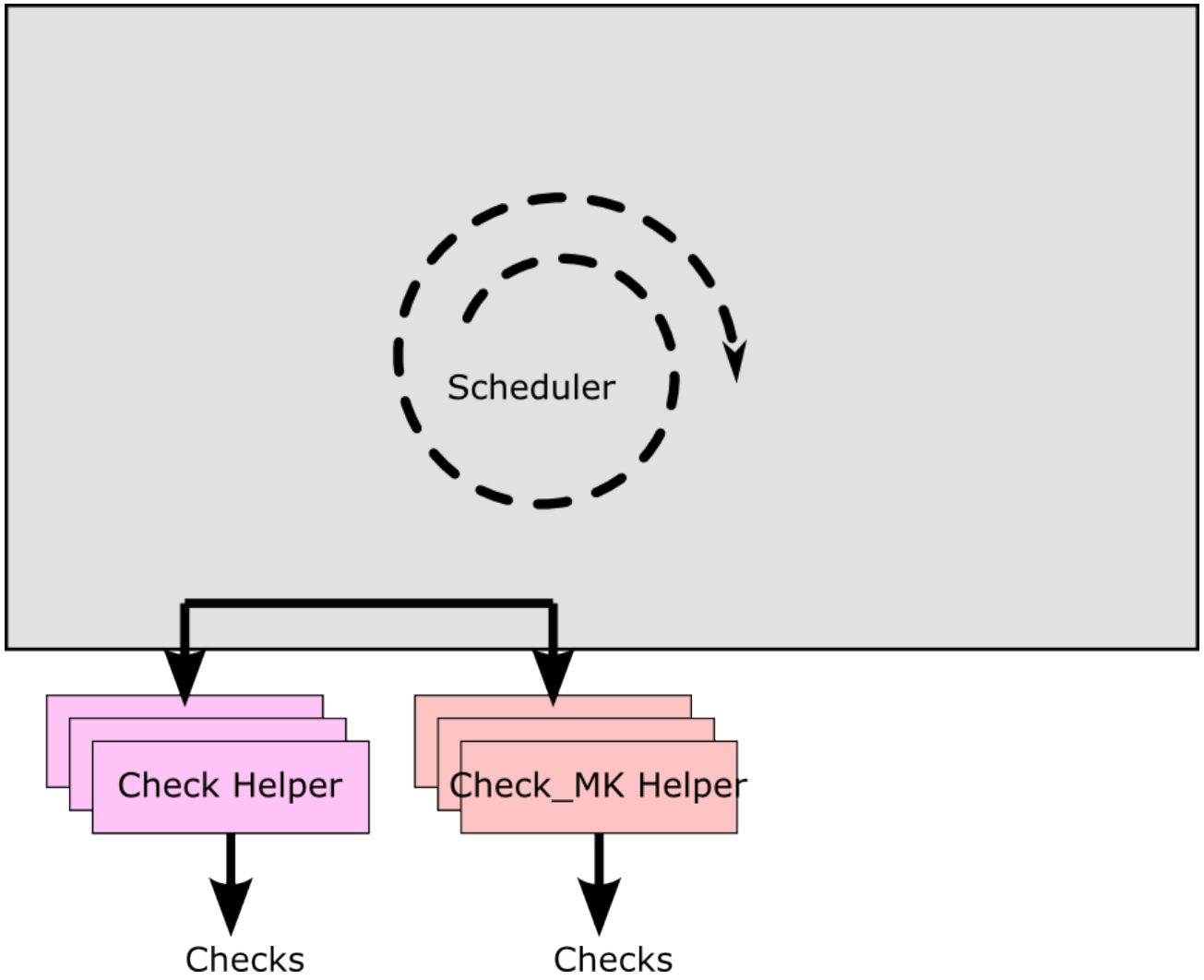




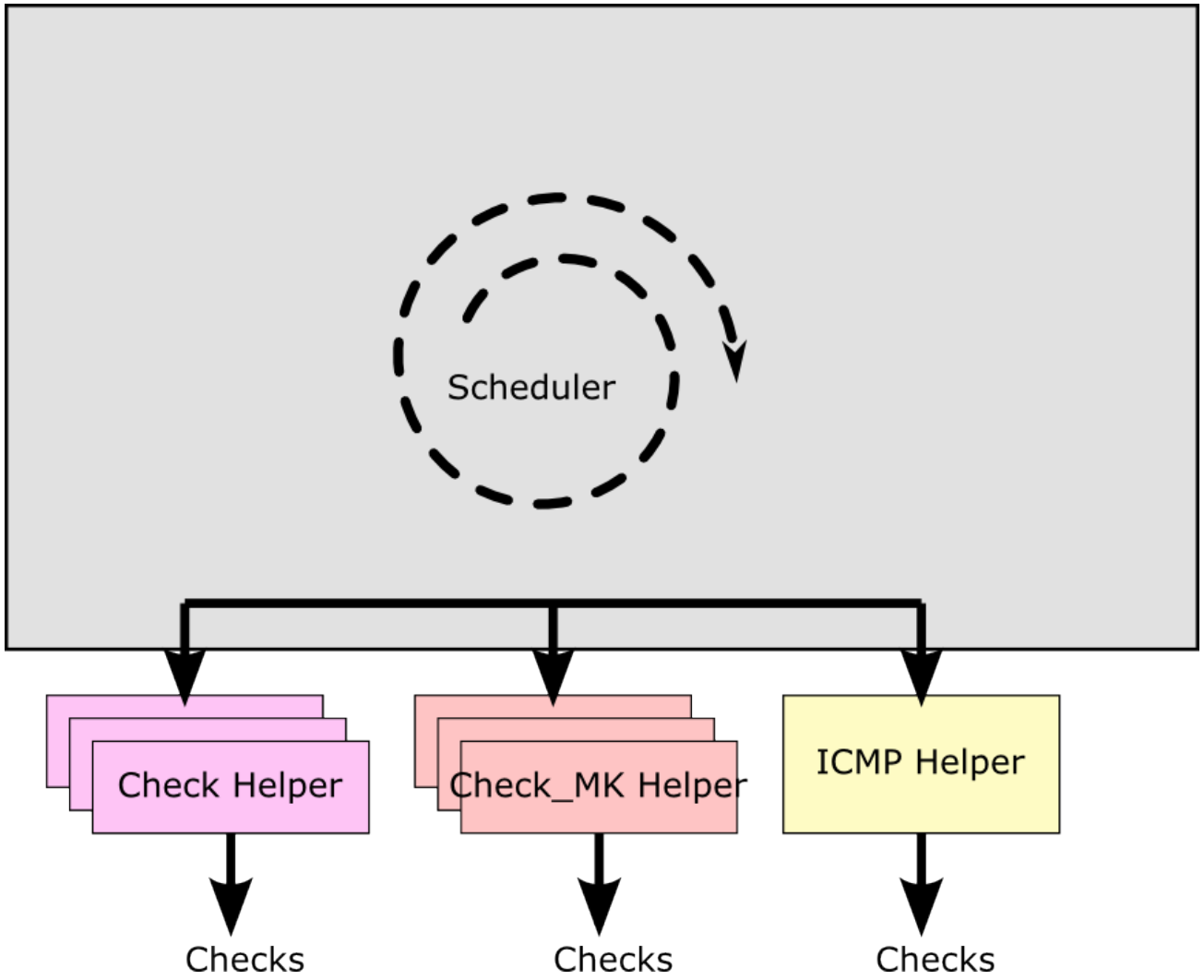
Architektur - Check Helper



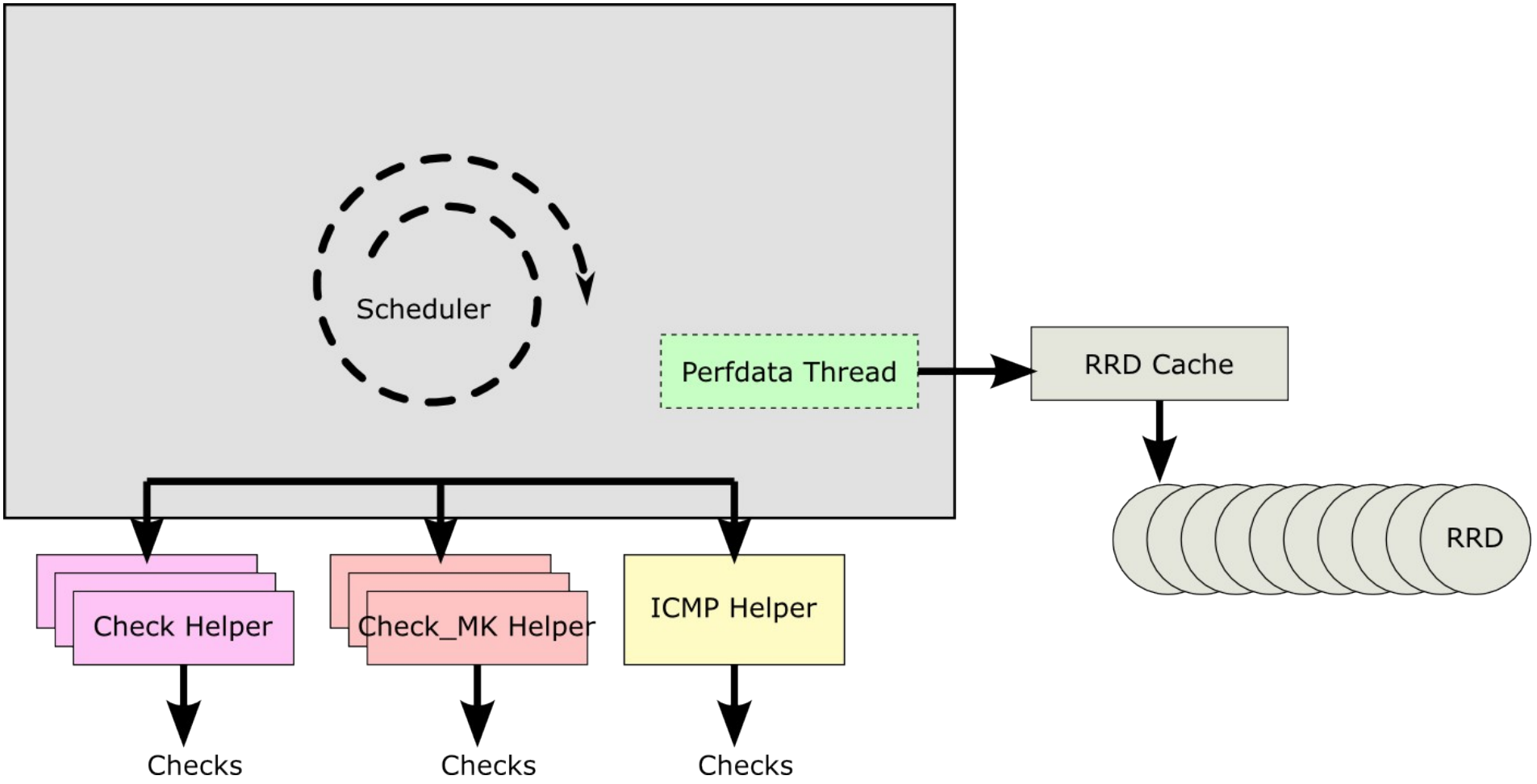
Architektur - Check_MK Helper



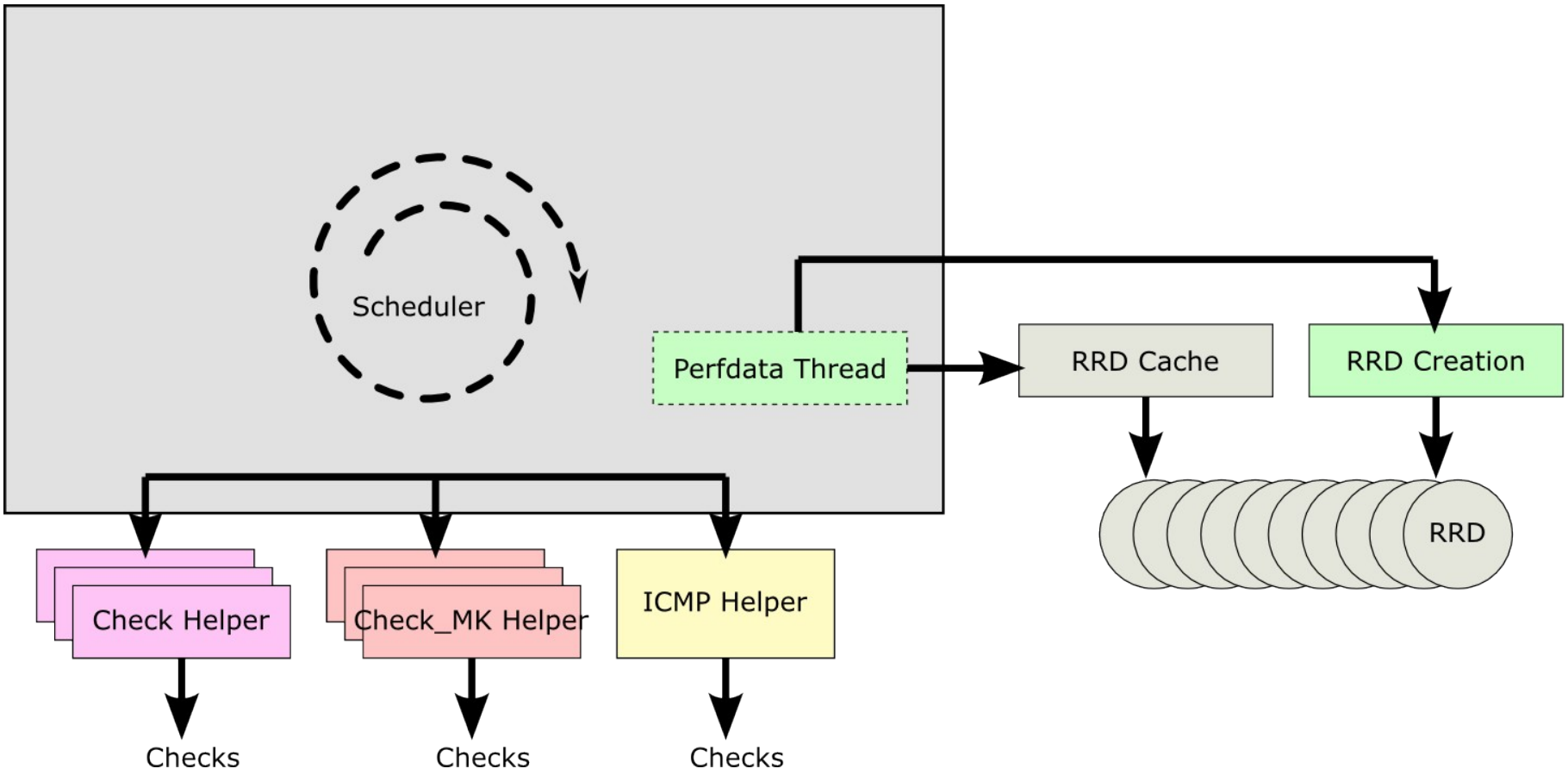
Architektur - ICMP Helper



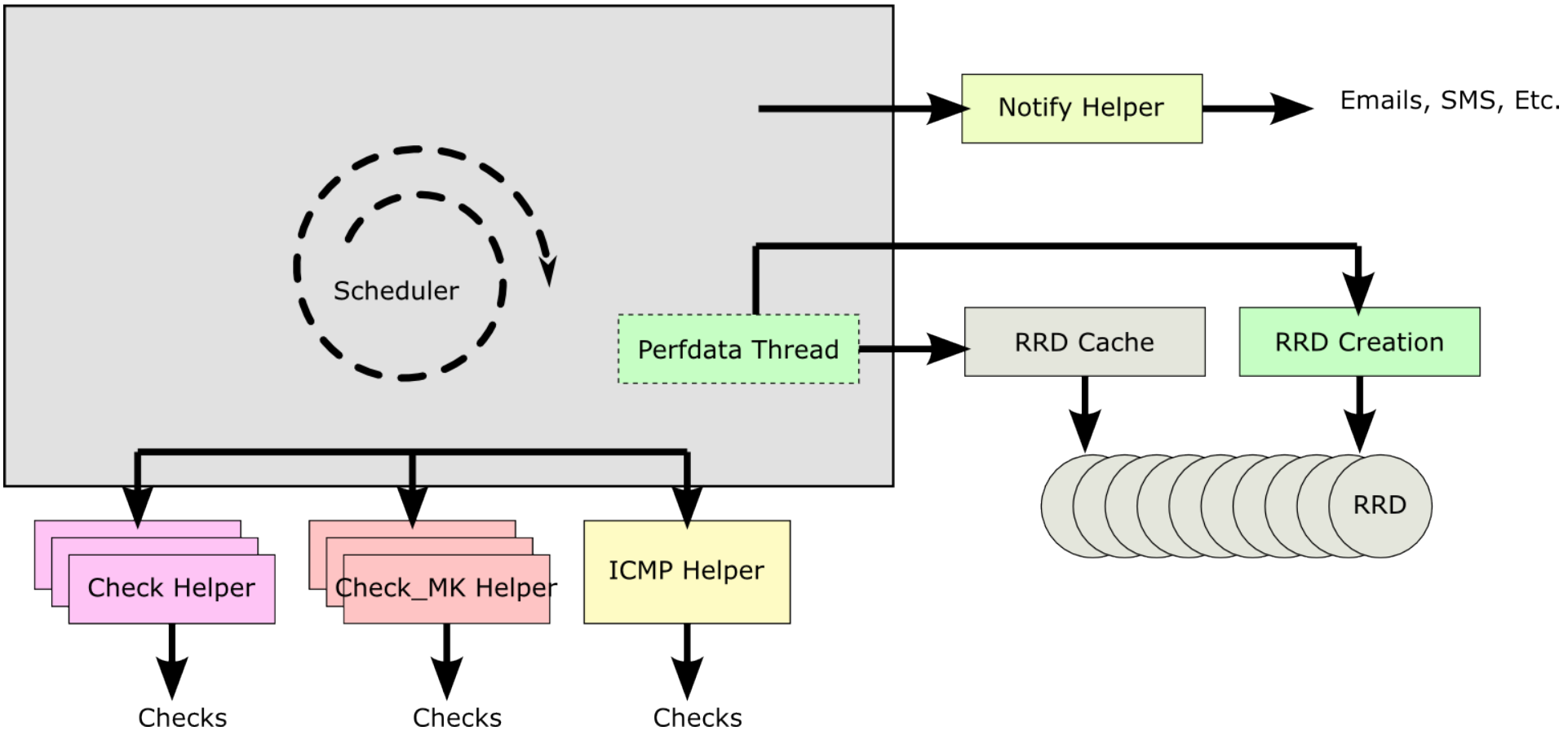
Architektur - RRD Cache

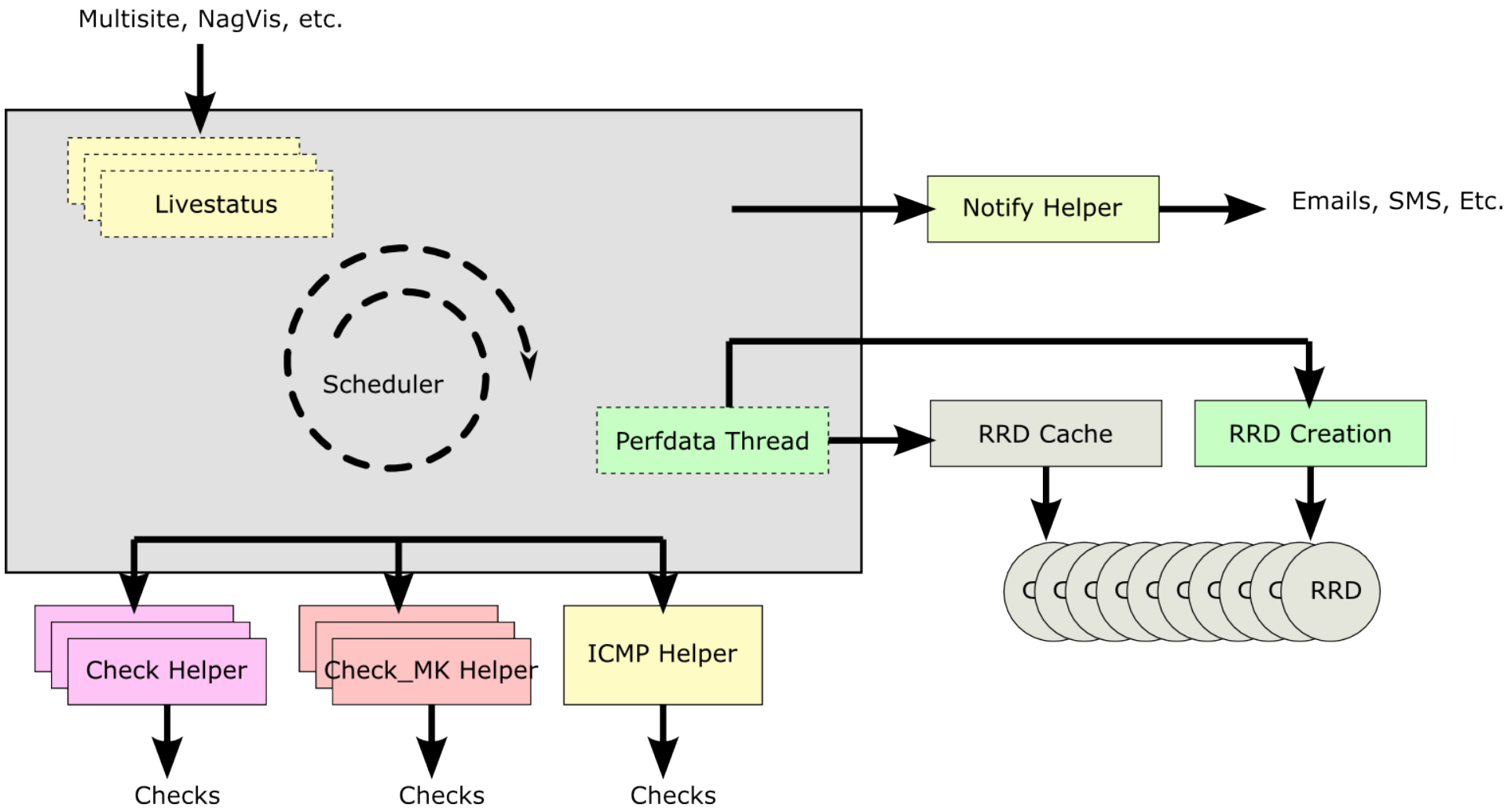


Architektur - RRD Erzeugung



Architektur - Notifications





Vielen Dank

für Ihre

Aufmerksamkeit