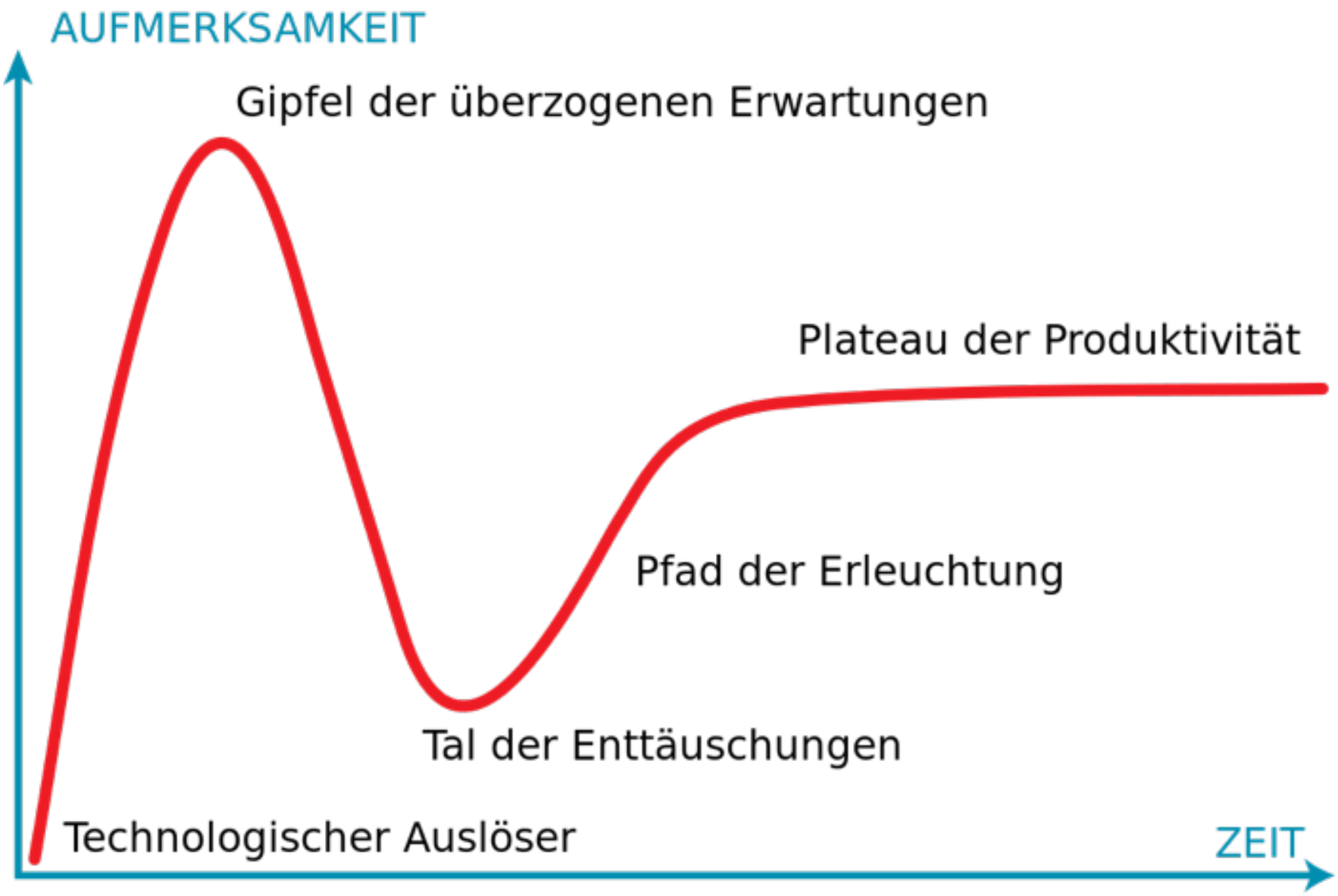
Performance synchroner und asynchroner I/O in Netzwerkanwendungen

parallel 2015

Hubert Schmid



Hype-Zyklus

Was ist von der Performance synchroner und asynchroner I/O in Netzwerkanwendungen zu erwarten?

Agenda

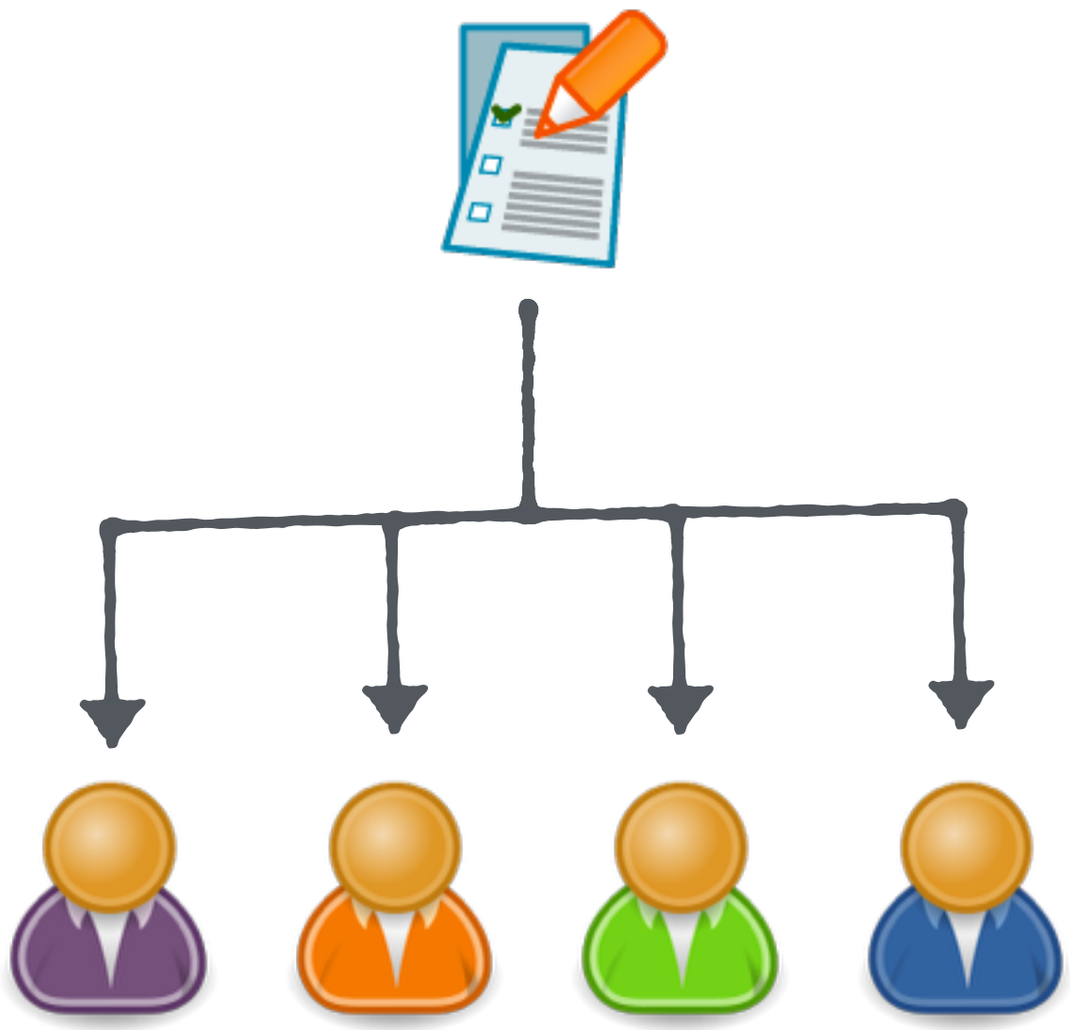
Einleitung

Beispiel, Benchmark, Ergebnisse

Stärken, Schwächen, Mischformen

Abschluss

Parallelität



Optimierung Durchlaufzeit

Nebenläufigkeit

Optimierung Effizienz

Terminologie

Synchrones Modell:

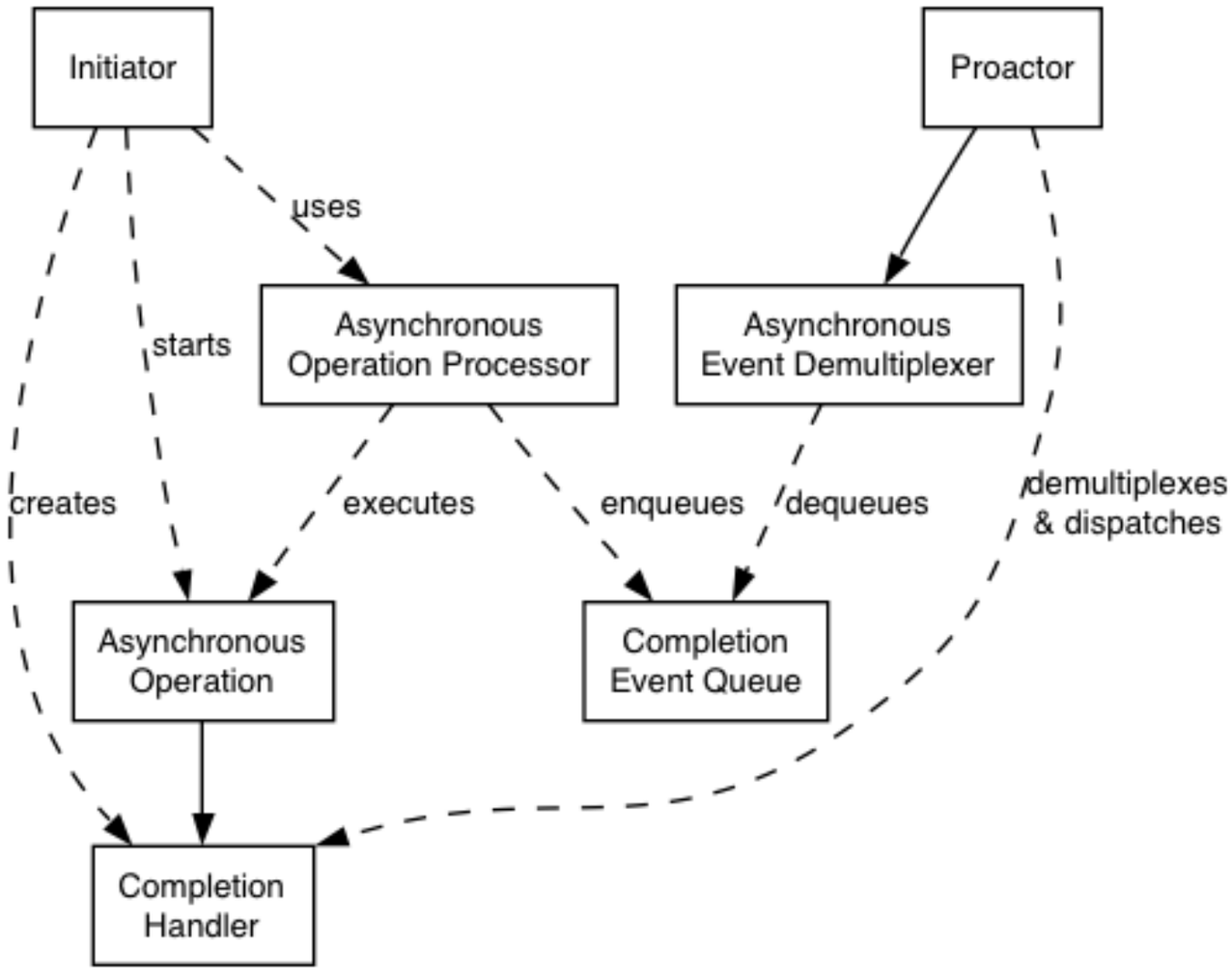
- blockierende I/O
- multi-threaded

Asynchrones Modell:

- nicht-blockierende I/O
- Proactor Pattern
  - notify-on-completion
  - notify-on-readiness

Abgrenzung:

- Asynchronität im Kernel, in der Hardware und im Netzwerk



Proactor Design Pattern
(Documentation of Boost.Asio 1.57)

Beispiel, Benchmark, Ergebnisse

Reverse-Echo-Server

- konkret
- einfach
- effektiv
- systemnah
- realitätsnah

Synchrone Variante

```
try {
    auto timeout = 300s;
    deadline deadline(timeout);
    while (auto n = _stream.getline(deadline)) {
        auto data = _stream.data();
        std::reverse(data, data + n - 1);
        _stream.write_n(data, n, deadline);
        _stream.drain(n);
        deadline.expires_from_now(timeout);
    }
    if (_stream.available())
        throw std::runtime_err
    }
} catch (...) {
    _handle_error();
}
```

```
::recv(_sock, ...);
if (errno == EAGAIN) {
    ::poll(_sock, _timer);
    ::recv(_sock, ...);
}
```

```
::send(_sock, ...);
if (errno == EAGAIN) {
    ::poll(_sock, _timer);
    ::send(_sock, ...);
}
```

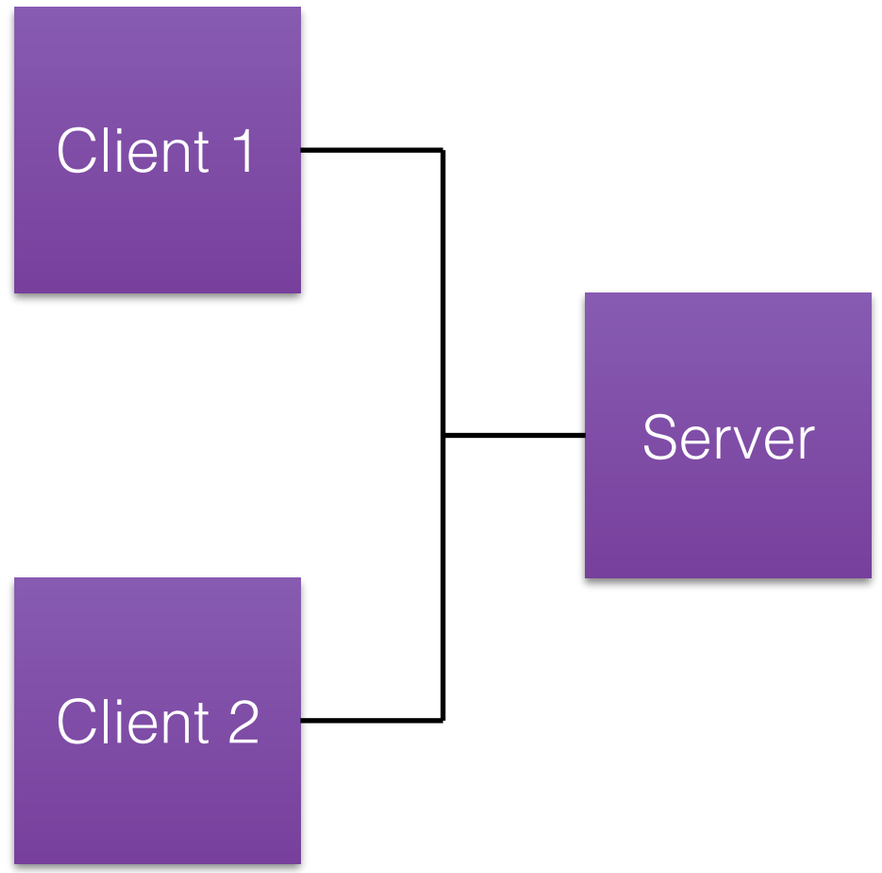
Asynchrone Variante

```
_stream.expires_from_now(300s, self);
_stream.async_getline(
    [this,self=std::move(self)](..., size_t n) {
        if (_stream.good(ec)) {
            auto data = _stream.data();
            std::reverse(data, data + n - 1);
            _stream.async_write_n(data, n,
                [this,self=std::move(self)](...) {
                    if (_stream.good(ec)) {
                        _stream.drain(n);
                        _async_run(std::move(self));
                    } else {
                        _handle_error(ec, "sending");
                    }
                });
        } else {
            _handle_error(ec, "receiving");
        }
    });
```

Testkandidaten und -aufbau

- Async Single-Core (Single-Threaded)
- Async Multi-Core (Thread pro Core)
- Sync Multi-Core (Thread pro Verbindung)

Messung durch zwei Systeme mit je 500.000 simultanen TCP-Verbindungen.



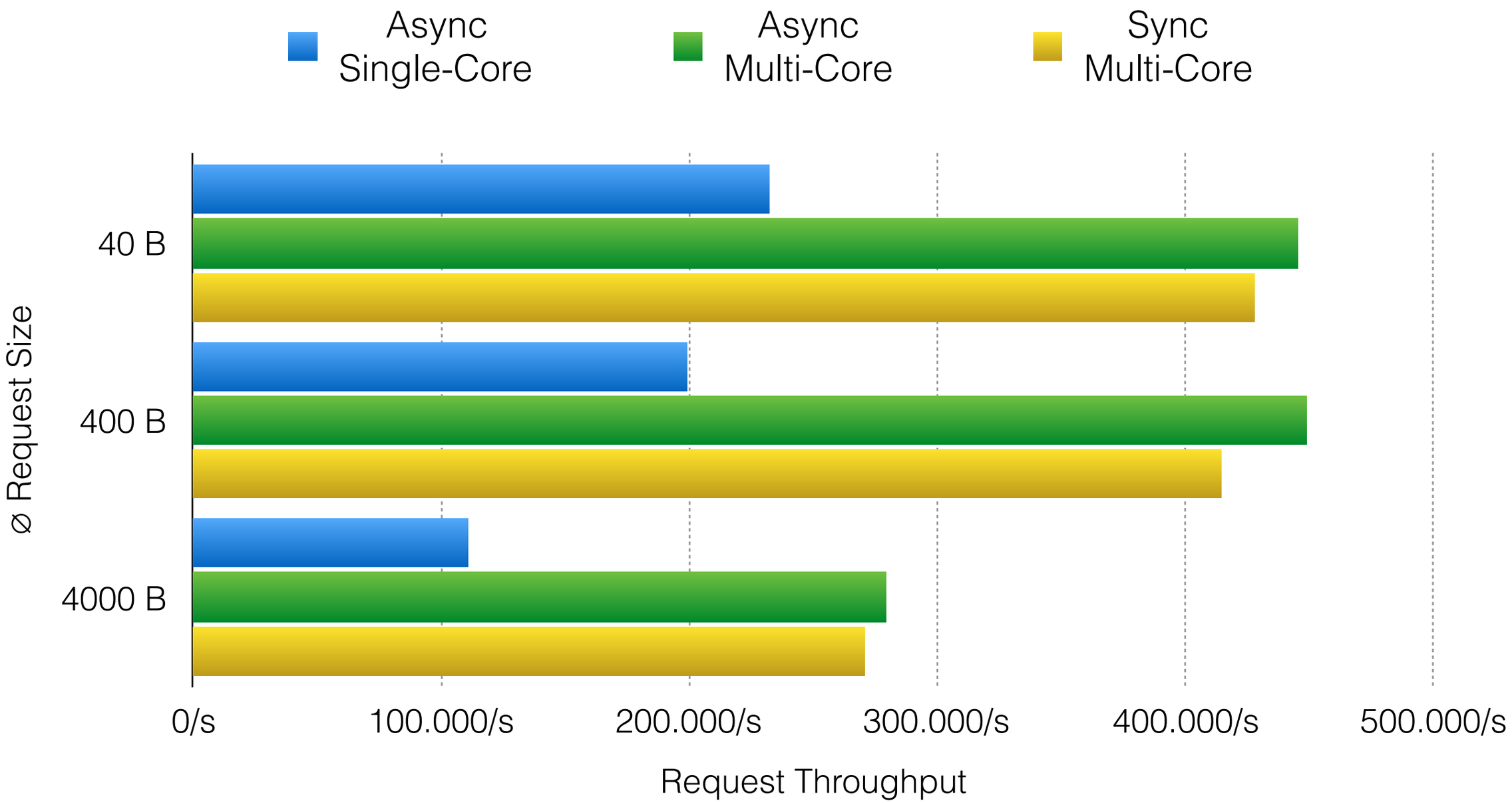
Systeme

- 2x Intel Xeon E5-2666 10C
- Intel NIC 10 GbE (2x Multi-Queue)
- Linux 3.16 (Debian jessie/testing)

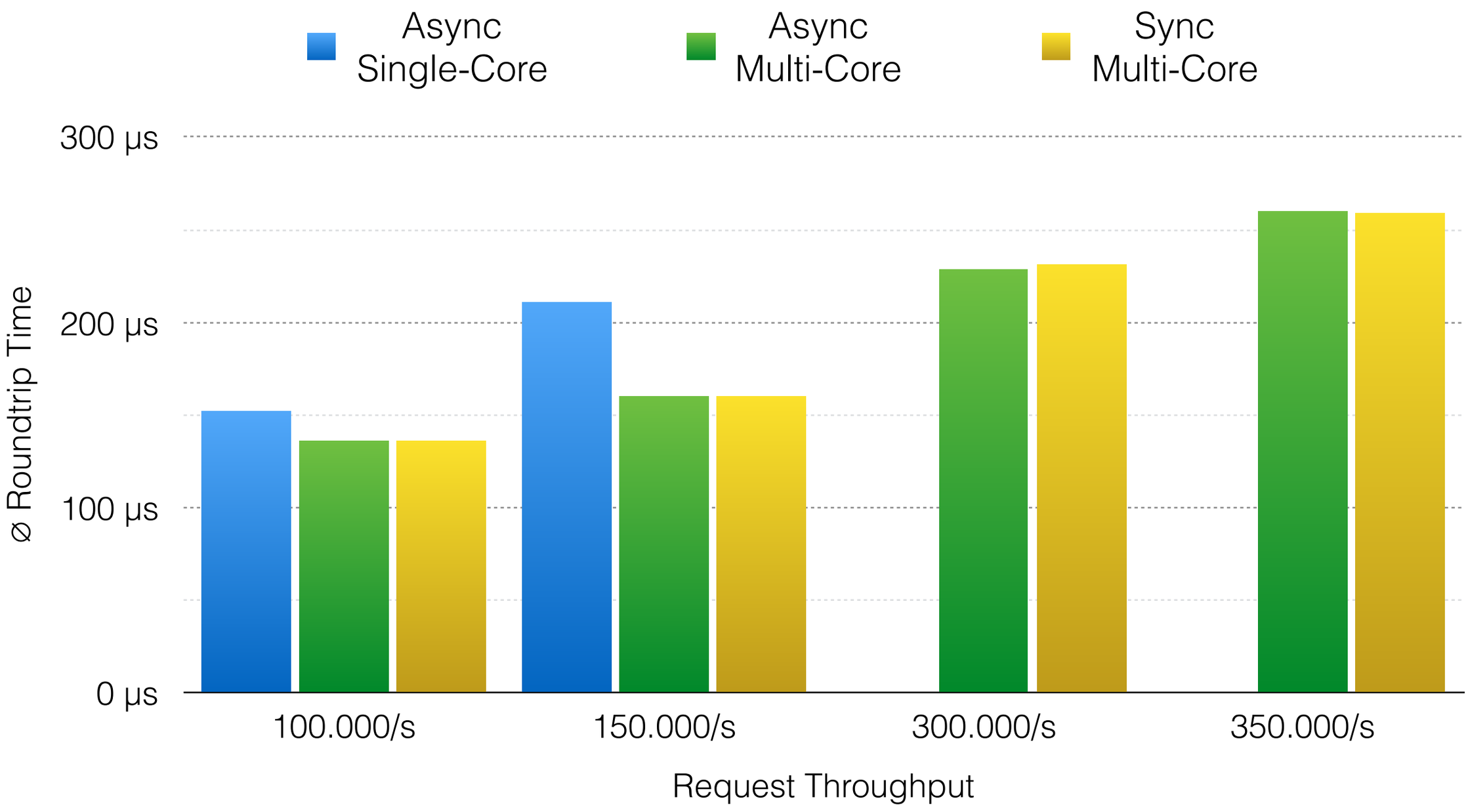
Netzwerk

- 10 Gb Ethernet (shared)

Durchsatz

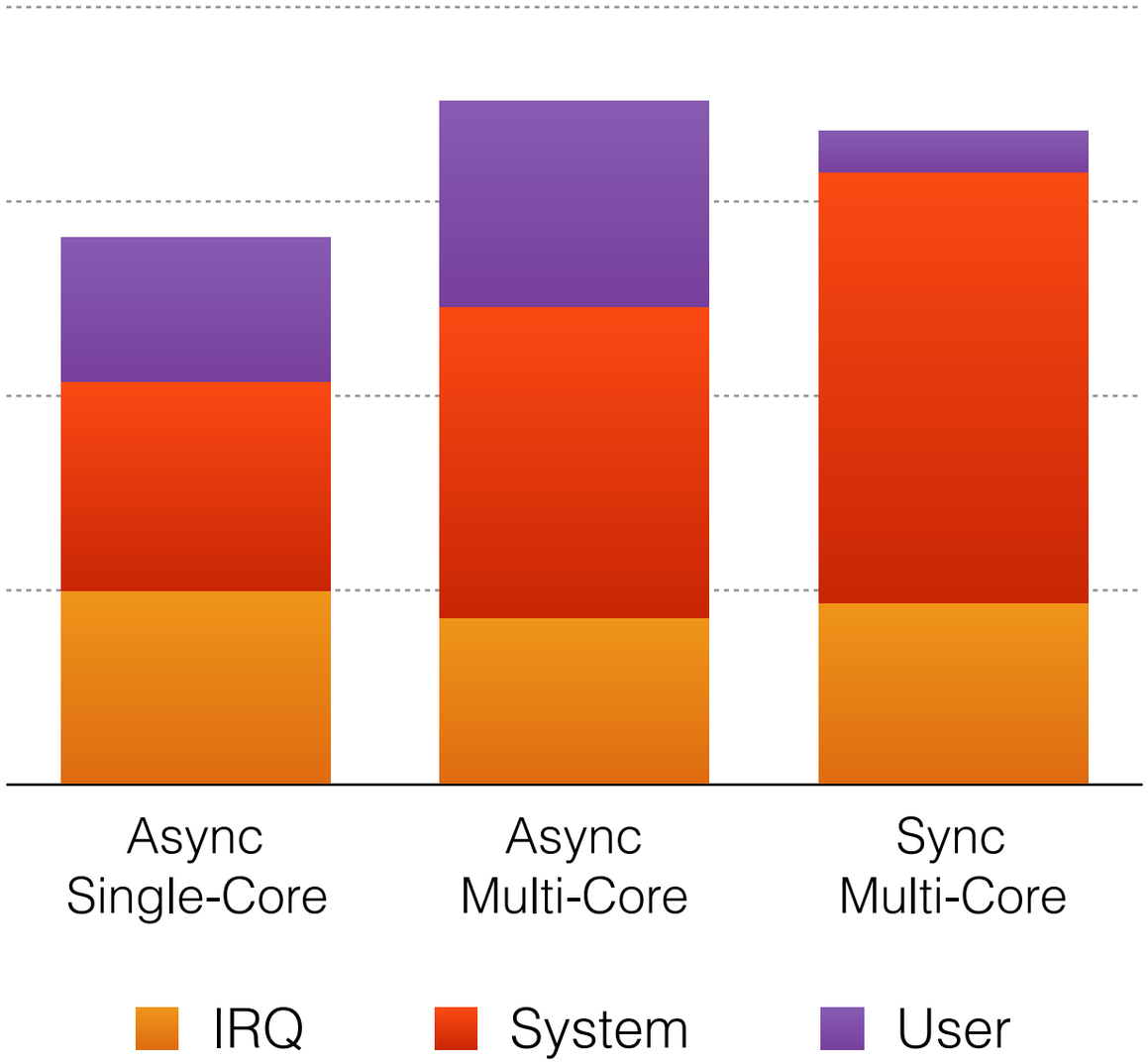

Async
Single-Core
Async
Multi-Core
Sync
Multi-Core
40 B
400 B
4000 B
Ø Request Size
0/s
100.000/s
200.000/s
300.000/s
400.000/s
500.000/s
Request Throughput

Latenz

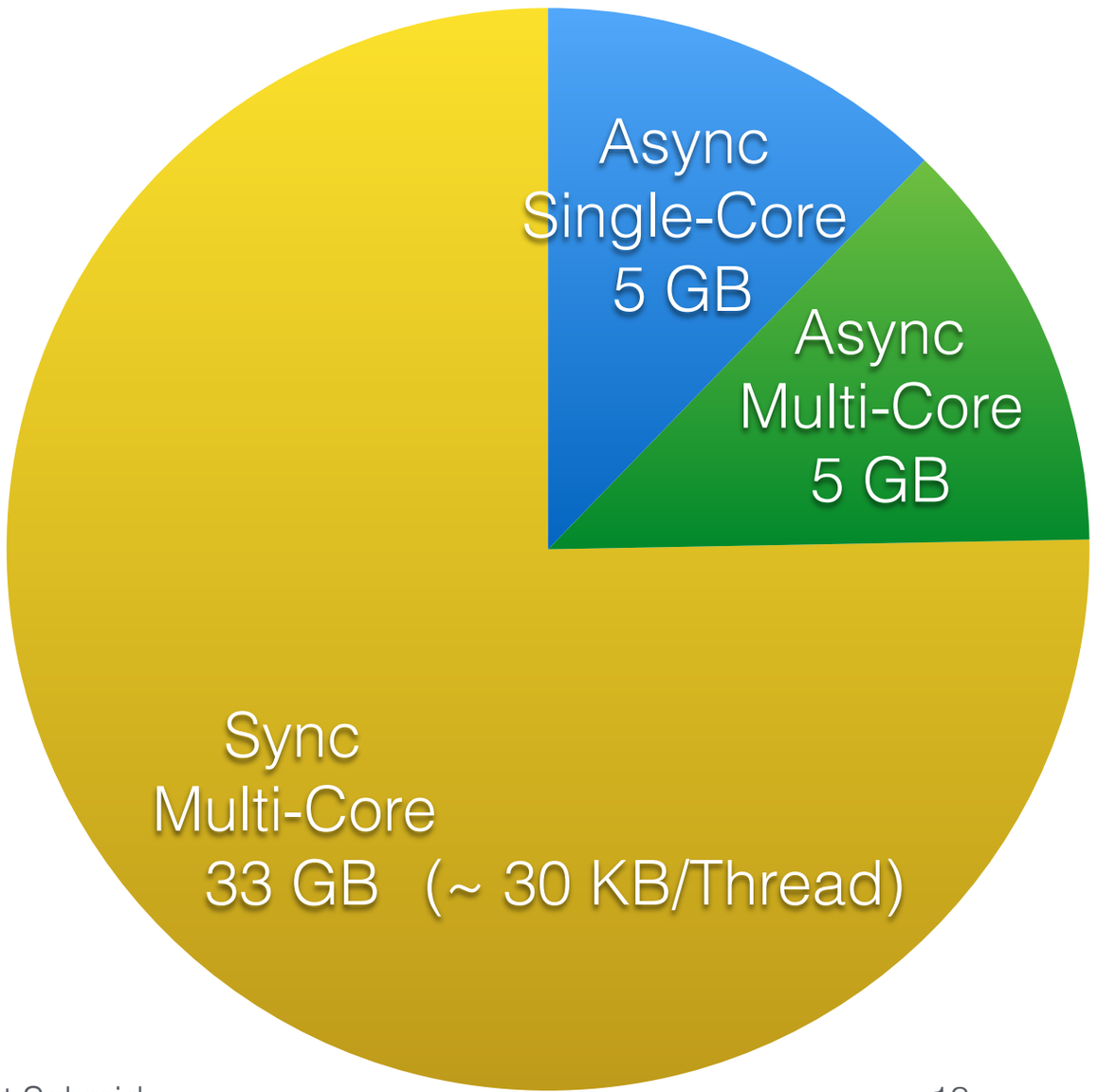

Async
Single-Core
Async
Multi-Core
Sync
Multi-Core
300 µs
200 µs
100 µs
0 µs
ø Roundtrip Time
100.000/s
150.000/s
300.000/s
350.000/s
Request Throughput

Ressourcen

CPU



Hauptspeicher



Synchron

```
try {
  auto timeout = 
  deadline deadline(timeout);
  while (auto n = _stream.getline
    aut a ream.data();
    std. erse data, data + n -
    _stream.write_n(data, n, dead
    _stream.drain(n);
    deadline.expires_from_now(tim
  }
  if (_ m.available()) {
    throw std::runtime_error("pro
  }
} catch (...) {
  _handle_error();
}
```

3-ε 1-ε 1+ε ε 1

Asynchron

```
_stream.expires_from_now(300s, se
_stream.async_getline(
  [this elf=std::move(self)](...
    if re od(ec)) {
      auto data = _stream.data();
      std::reverse(data, data + n
      _stream.async_write_n(data,
        this elf=std::move(self
        eam.good(ec)) {
          _stream.drain(n);
          _async_run(std::move(
        } else {
          _handle_error(ec, "se
        }
      });
  } else {
    _handle_error(ec, "receivin
  }
```

1 2+ε 1 1+ε 1

Systemaufrufe

Thread Kontextwechsel

Task Kontextwechsel

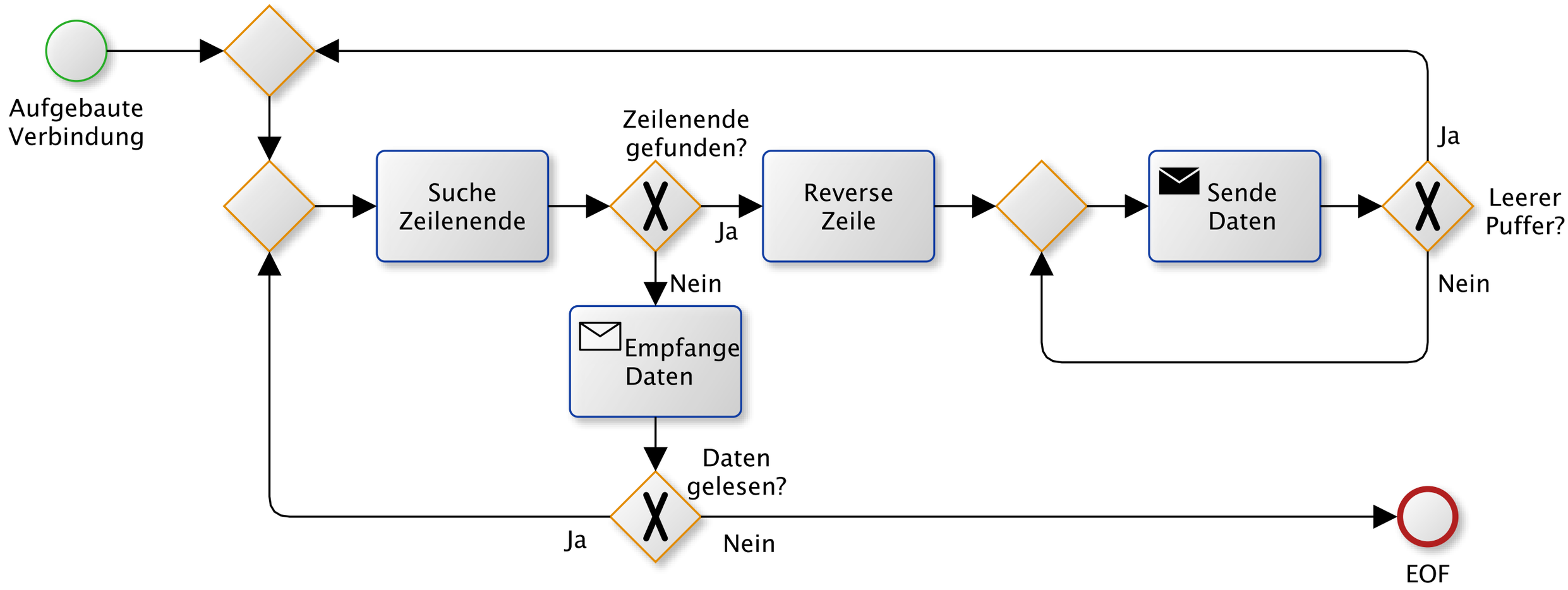
Ergebnisse

- Saturation durch Multi-Core
- Ähnliche Abläufe im Netzwerk-Stack
- Unterschiede beim Hauptspeicher-Bedarf
- Optimierung unabhängig vom Programmiermodell

Stärken, Schwächen, Mischformen

Asynchrone Stärken

- Plattform-Unterstützung
- Kernel-Bypass
- Spezialisierung
- Nicht-sequentielle Abläufe

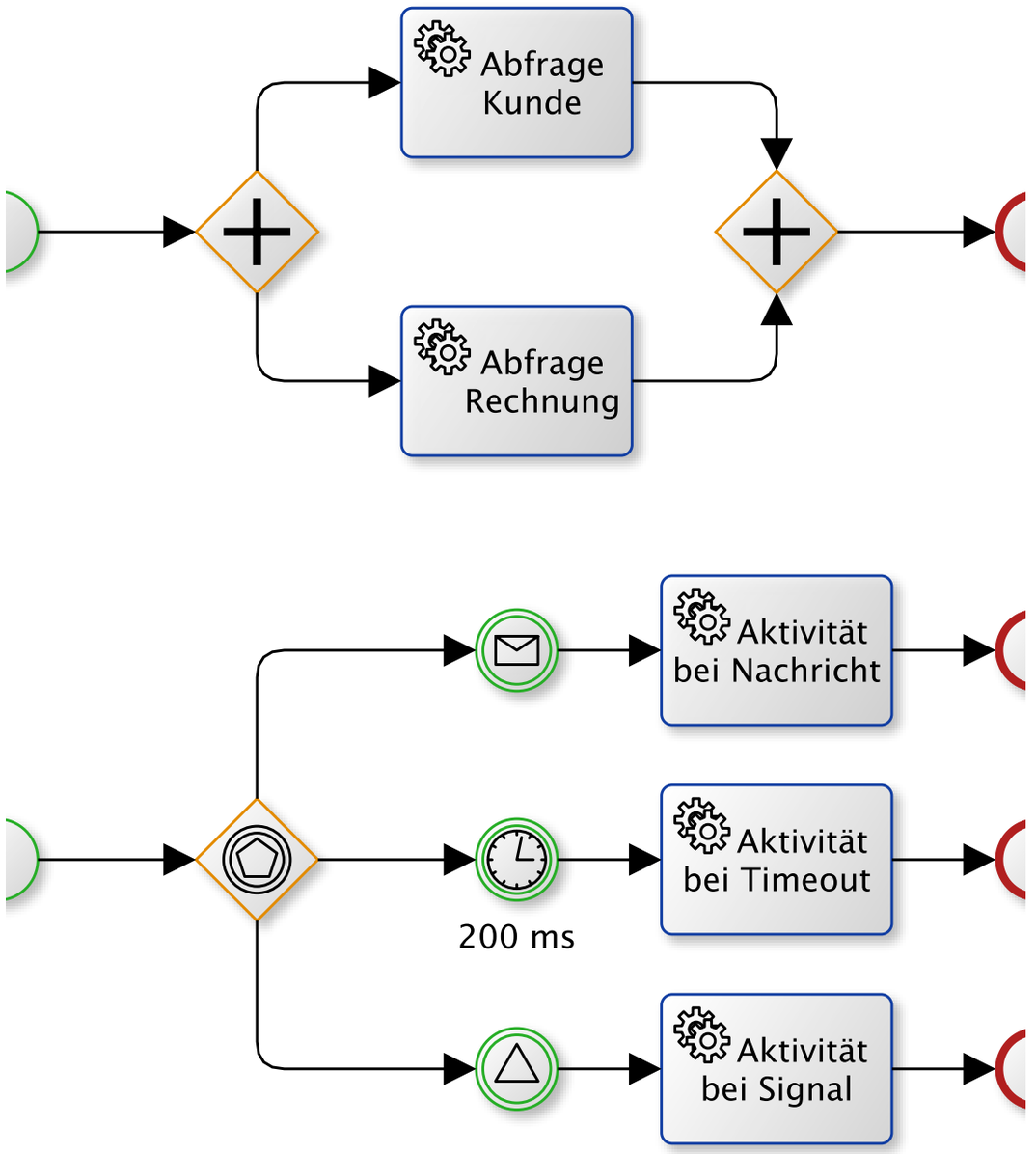


Ablaufdiagramm

Rein sequentieller Programmablauf
beim Reverse-Echo-Server

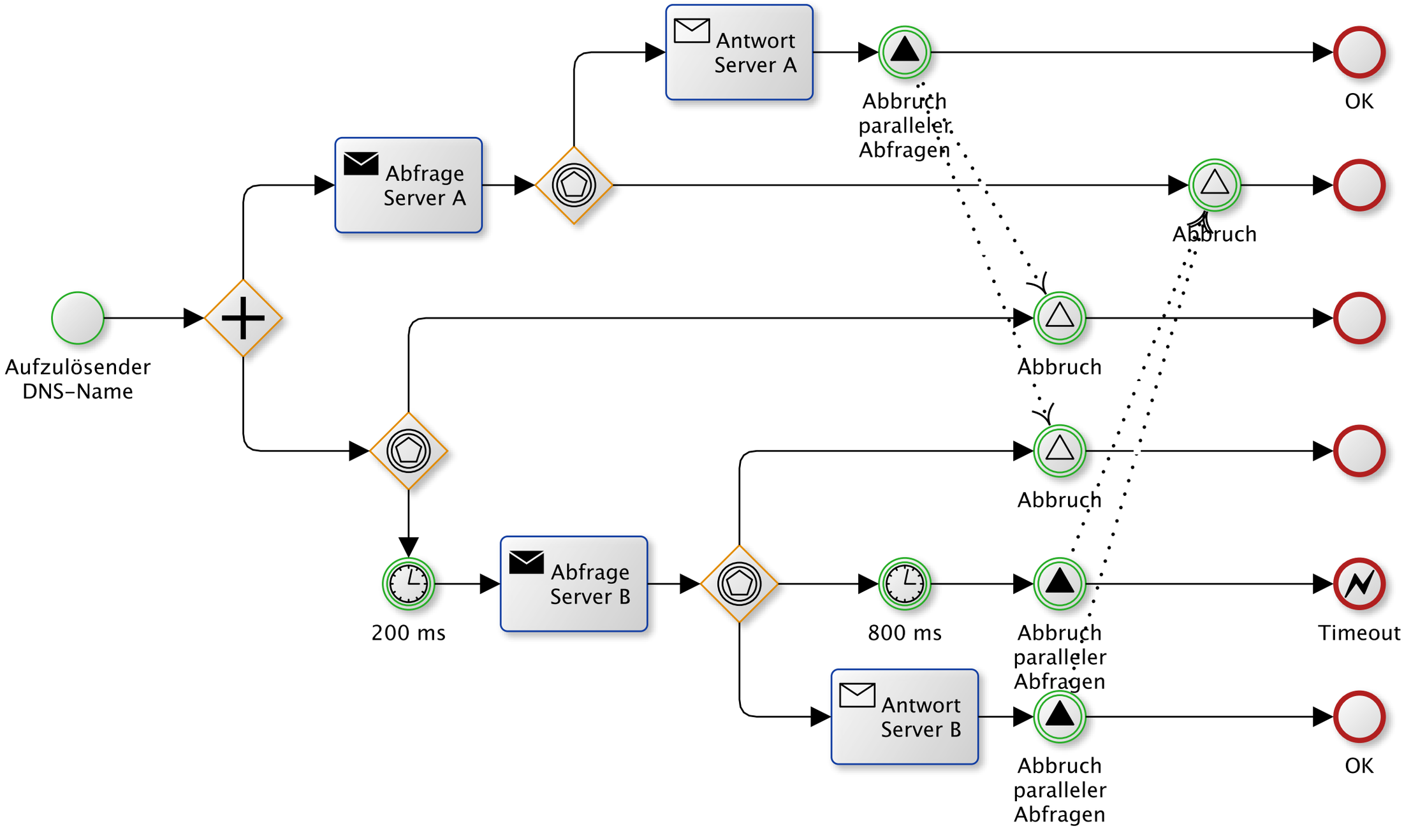
Nicht-sequentielle Abläufe

- Parallele Verzweigung (Overlapping I/O)



- Ereignis-basierte Entscheidungspunkte
- Steuerung paralleler Ablaufpfade

Beispiel DNS-Abfrage



Asynchrone Schwächen

Betriebssystem

- unvollständige Schnittstelle (Kernel, Kern-Bibliotheken)
- virtueller Speicher

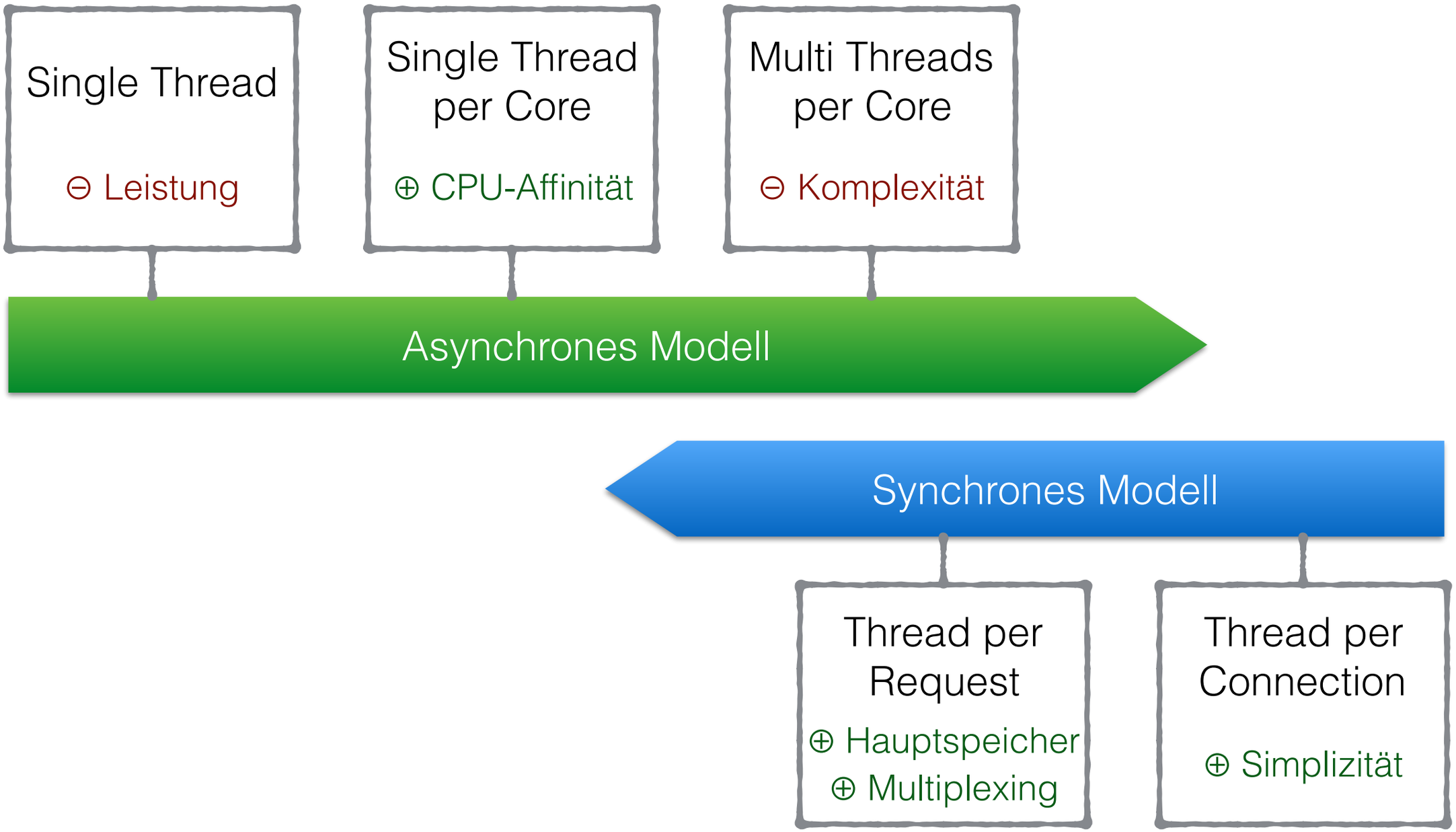
Rechenintensive Abläufe

- keine Präemption (Latenz)

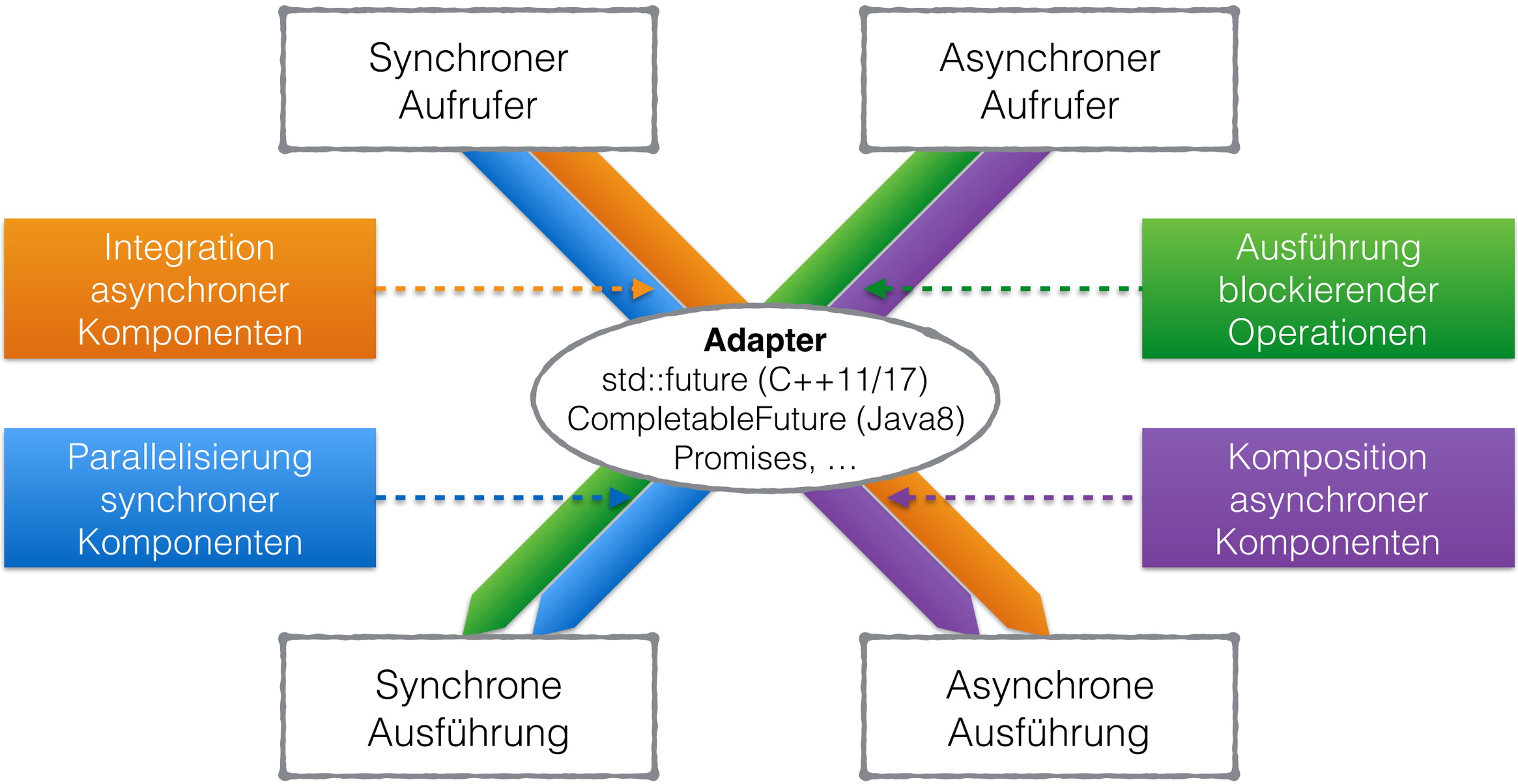
Fremdbibliotheken

- unvollständig, inkompatibel

Mitigation


Single Thread
⊖ Leistung
Single Thread per Core
⊕ CPU-Affinität
Multi Threads per Core
⊖ Komplexität
Asynchrones Modell
Synchrones Modell
Thread per Request
⊕ Hauptspeicher
⊕ Multiplexing
Thread per Connection
⊕ Simplizität

Kombination



Zusammenfassung

- Multi-Core
- Hauptspeicher und Plattform-Spezifika
- Nicht-sequentielle Abläufe
- Blockierende Operationen
- Mischformen

Quellen

- “The C10K problem” by Dan Kegel
- “Scaling in the Linux Networking Stack” by Tom Herbert and Willem de Bruijn
- “Comparing the performance of web server architectures.” by Pariag, David, et el.
- “Hype-Zyklus nach Gartner Inc.” by “Idotter” is licensed under CC BY-SA 3.0
- “Arcadia pocket watch 1859” by “Suebun” is licensed under CC BY-SA 3.0
- “An icon in the Tango! theme style: Blue person. Green person. Orange person. Purple person.” by “Inductiveload” is released into public domain.
- “Emblem persons blue green” by “Con-struct” is licensed under CC BY-SA 3.0
- “View refresh” by “The people from the Tango! project” is released into public domain.
- “Korganizer Icon” by “Umut Pulat” is licensed under GNU Lesser General Public License
- “Proactor design pattern” by “Christopher M. Kohlhoff” is licensed under Boost Software License 1.0
- “AE.svg” by “C.Thure” is released into public domain
- “Question mark made of question marks” by “Khaydock” is licensed under CC BY-SA 3.0