Das C++ Actor Framework (CAF) im Leistungsvergleich

Marian Triebe, Dominik Charousset, Raphael Hiesgen,Thomas C. Schmidt
{marian.triebe, dominik.charousset, raphael.hiesgen, t.schmidt}@haw-hamburg.de

iNET RG, Department of Computer Science
Hamburg University of Applied Sciences

11. September 2015

Hochschule für Angewandte Wissenschaften Hamburg
Hamburg University of Applied Sciences

Agenda

1. Einleitung
   - Motivation
   - Aktorenmodell
   - Ziele von CAF
2. Hintergrund
   - Vorstellung C++ Actor Framework (CAF)
   - Vergleichbare Frameworks
3. Evaluierung
   - Aktorimplementierungen
   - High-Level (CAF) vs. Low-Level (MPI)
   - Transparente Integration von GPUs
4. Fazit & Ausblick

Agenda

Motivation

- Verteilte Systeme & deren Entwicklung
- Lokale Nebenläufigkeit

Aktorenmodell

- Isolierte, nebenläufige Entitäten: Aktoren
- Verteilungstransparent
- Dynamisches Starten von Aktoren
- Fehlermodell

Aktorenmodell

Der Aktor

- Kann auf Erhalt von Nachrichten reagieren:
  - Neue Nachrichten verschicken
  - Neue Aktoren erstellen (”spawnen”)
  - Sein Verhalten ändern
- Ein Aktor kann nur eine Nachricht zur Zeit verarbeiten
- Ist in sich selbst sequentiell

Ziele von CAF

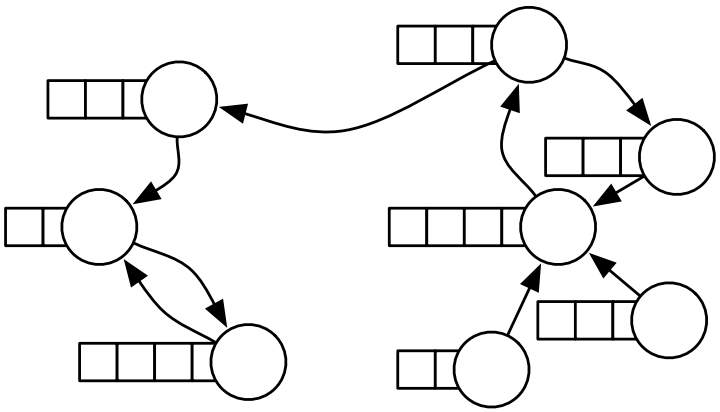
- Hochstehendes Programmiermodell
  - Effizienz (CPU-Profil, Speicherverbrauch)
  - Einsatz in Infrastruktur-Software
  - Vereinfachung beim Entwickeln von verteilten Systemen
- Einbindung von heterogener Hardware
  - Vermeidung von zusätzlichen Laufzeitkosten
  - Transparentes Ansprechen von Hardware

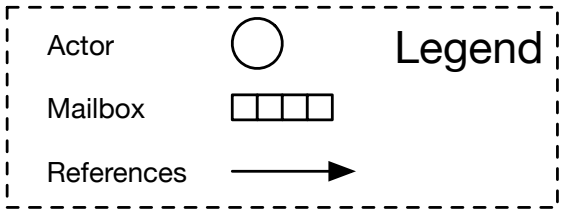
Agenda

Vorstellung CAF

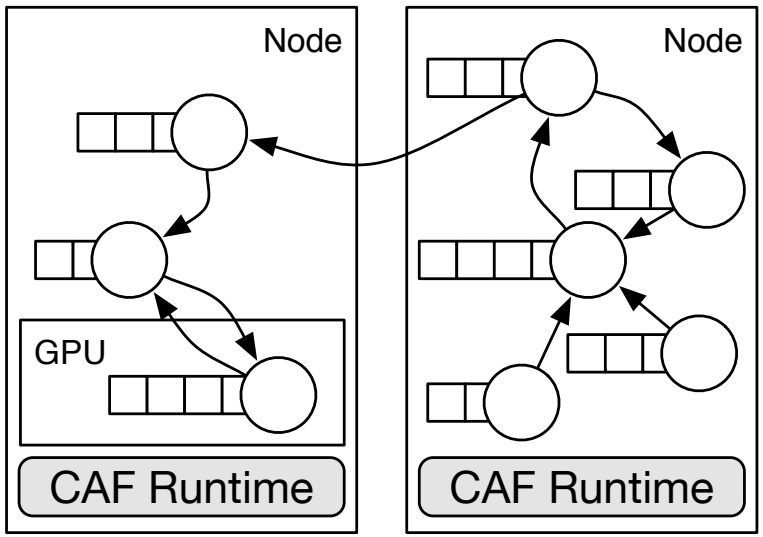
- Aktive Community
  - Github-Statistik (113 Forks, 626 Sterne, 23 Kontributoren)
  - Organisiert über Chat & Mailingliste
- Modularer Aufbau:
  - Core
  - IO
  - OpenCL
- Low-End Devices
- Unterstützt:
  - Linux
  - OSX
  - (Windows)
  - (iOS)
  - (Android)

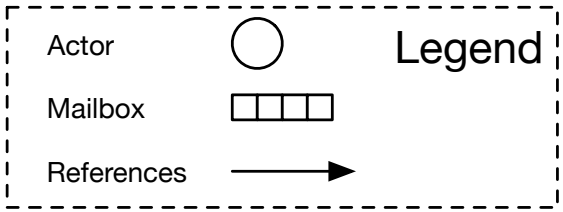
Architektur



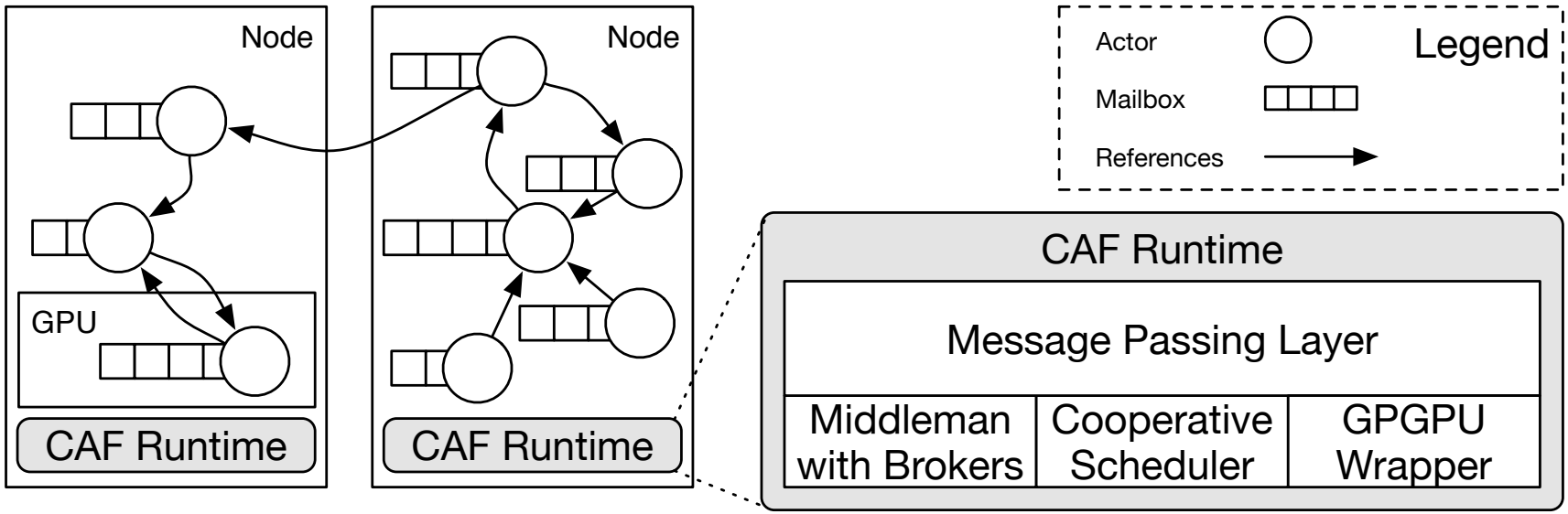


Architektur

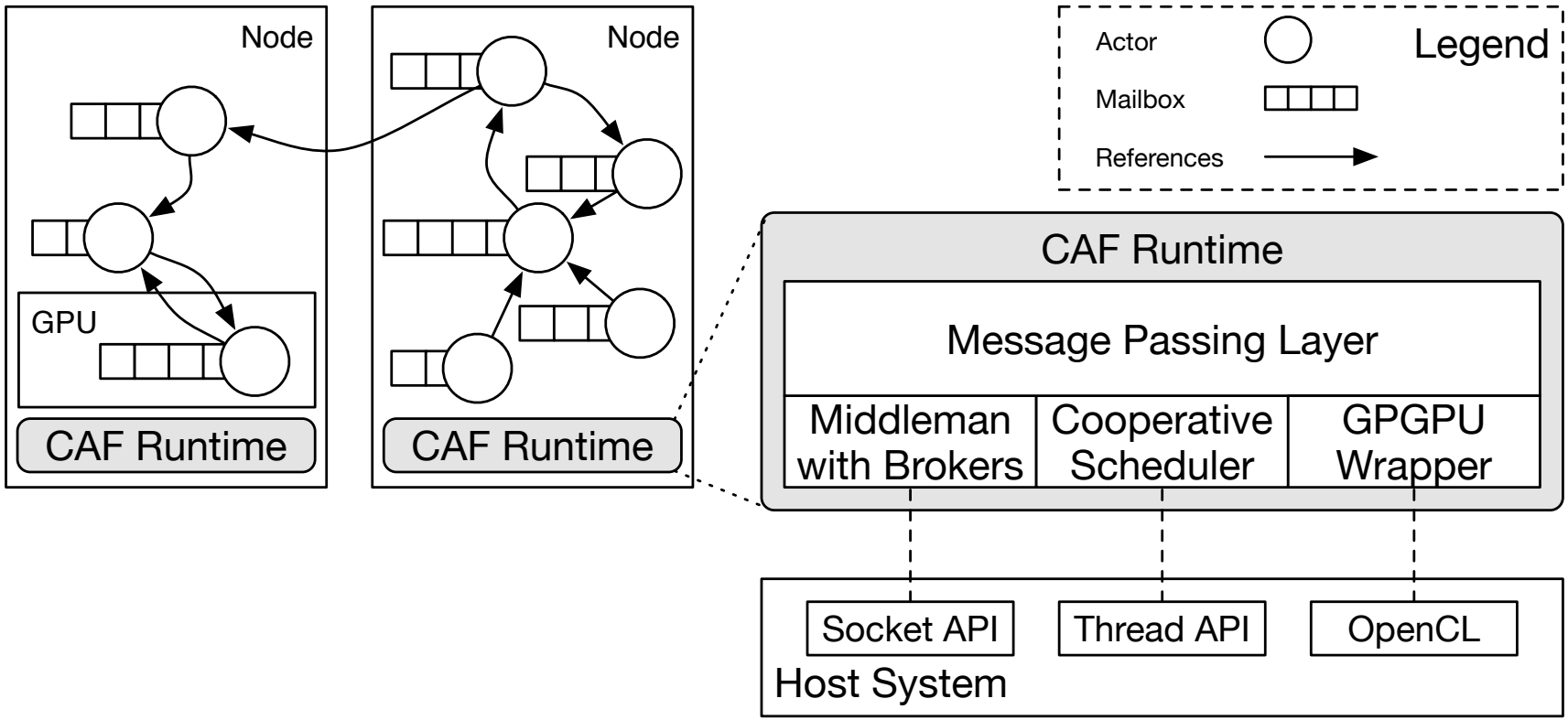




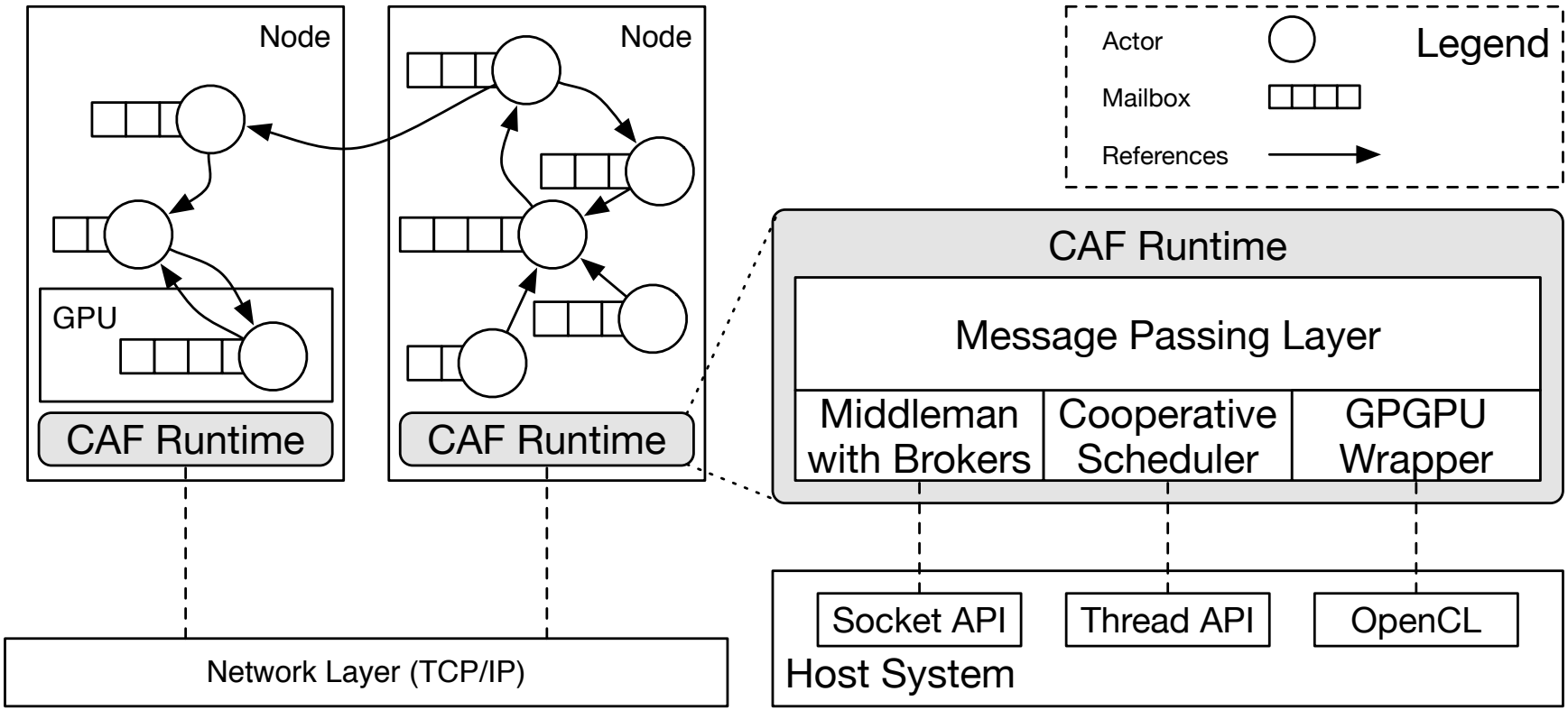
Architektur



Architektur



Architektur



Kritische Komponenten

Laufzeitkritische Komponenten:

- Kooperativer Scheduler
- Mailbox
- Transparentes Messaging im Netzwerk
- GPGPU-Wrapper

Kooperativer Scheduler

- Herausforderung:
  - Scheduling überhalb des Betriebssystems
  - Effiziente Verteilung von Aktoren auf Threads ($n : m$)
- Lösung:
  - Work-Stealing
    - P Worker, P Job-Queues
    - Wenn Job-Queue leer: "stehlen"
  - Aktoren müssen "fair" sein, oder vom Scheduling ausgeschlossen werden

Mailbox

- Herausforderung:
  - Mailbox ist Synchronisationspunkt
  - Sender und Empfänger dürfen sich nicht blockieren
- Lösung:
  - "Single-Reader-Many-Writer-Queue"
    - Nur Besitzer der Mailbox darf lesen
    - Alle dürfen schreiben

Transparentes Messaging
im Netzwerk

- Herausforderung:
  - Transparenter Zugriff auf entfernt laufende Aktoren
  - Netzwerk auslasten
  - Laufzeitverlust durch I/O vermeiden
- Lösung:
  - Mehrere CAF-Instanzen bilden ein einziges, verteiltes System

GPGPU-Wrapper

- Herausforderung:
  - Transparentes Ansprechen heterogener Hardware (z.B. GPU)
  - Möglichst geringer Laufzeitverlust
- Lösung:
  - Proxies für OpenCL-Kernel (”Actor Facade”)
  - OpenCL Modul in CAF verwaltet OpenCL Primitive

Verglichene Frameworks

- Erlang
  - Veröffentlicht: 1987
  - Ericsson
  - Erste de-fakto Umsetzung des Aktormodells
- JVM-Sprachen:
  - SalsaLite
    - Veröffentlicht: 2001
    - University of North Dakota
    - Eigene Sprache
  - Scala (Akka)
    - Veröffentlicht: 2003
    - Entwickelt an der École polytechnique fédérale de Lausanne (EPFL)
    - Aktoren stehen durch AKKA zur Verfügung
  - ActorFoundry
    - Veröffentlicht: zwischen 1998 & 2000
    - Entwickelt an der University of Illinois
    - Bytecode Transformation (Killim)

Verglichene Frameworks

- Nativ:
  - Charm++
    - Veröffentlicht: späte 80iger
    - Entwickelt an der University of Illinois
    - High Performance Computing
  - MPI
    - Veröffentlicht: 5. Mai 1994 (Version 1.0)
    - Argonne National Laboratory (ANL)
    - Standardisierte API
    - Hauptzielgebiet: High Performance Computing
    - Mehrere Implementierungen

Agenda

Szenario 1

- Verifizierung der laufzeitkritischen Komponenten
- Vergleich mit ähnlichen Frameworks
- Keine Verteilung, lokale Nebenläufigkeit

Testaufbau 1

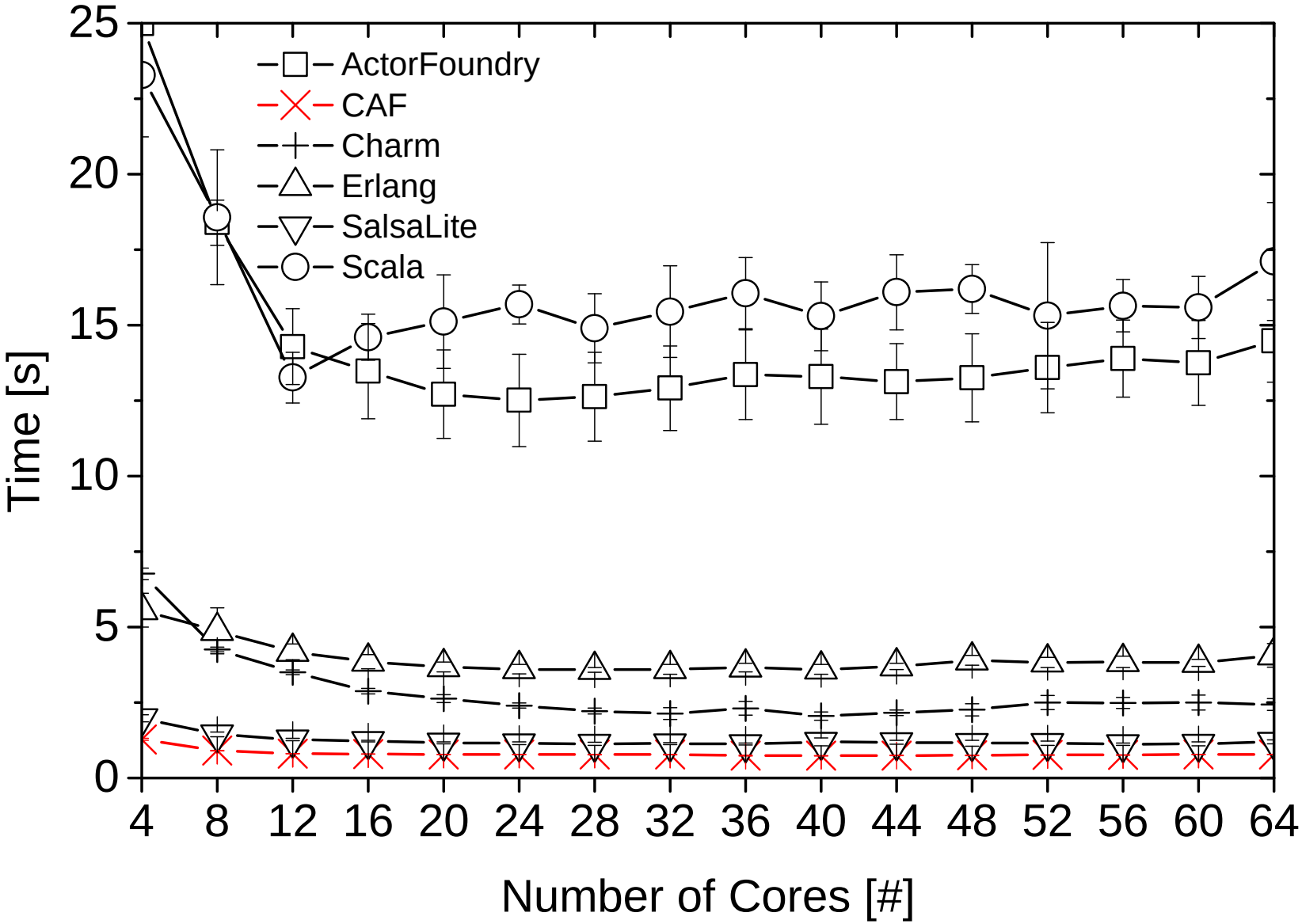
- Gemessen wurden Laufzeit & Speicherverbrauch
- Hostsystem:
  - Linux
  - Vier Opterons mit jeweils 16 Kernen
  - 512GB RAM

Divide & Conquer

- Wie effizient ist die Erzeugung von Aktoren?
- Problematische Komponente: Scheduler
- 2^{20} Aktoren rekursiv starten
- Erwartung:
  - Abfall der Laufzeit
  - Freien Speicher erneut verwenden

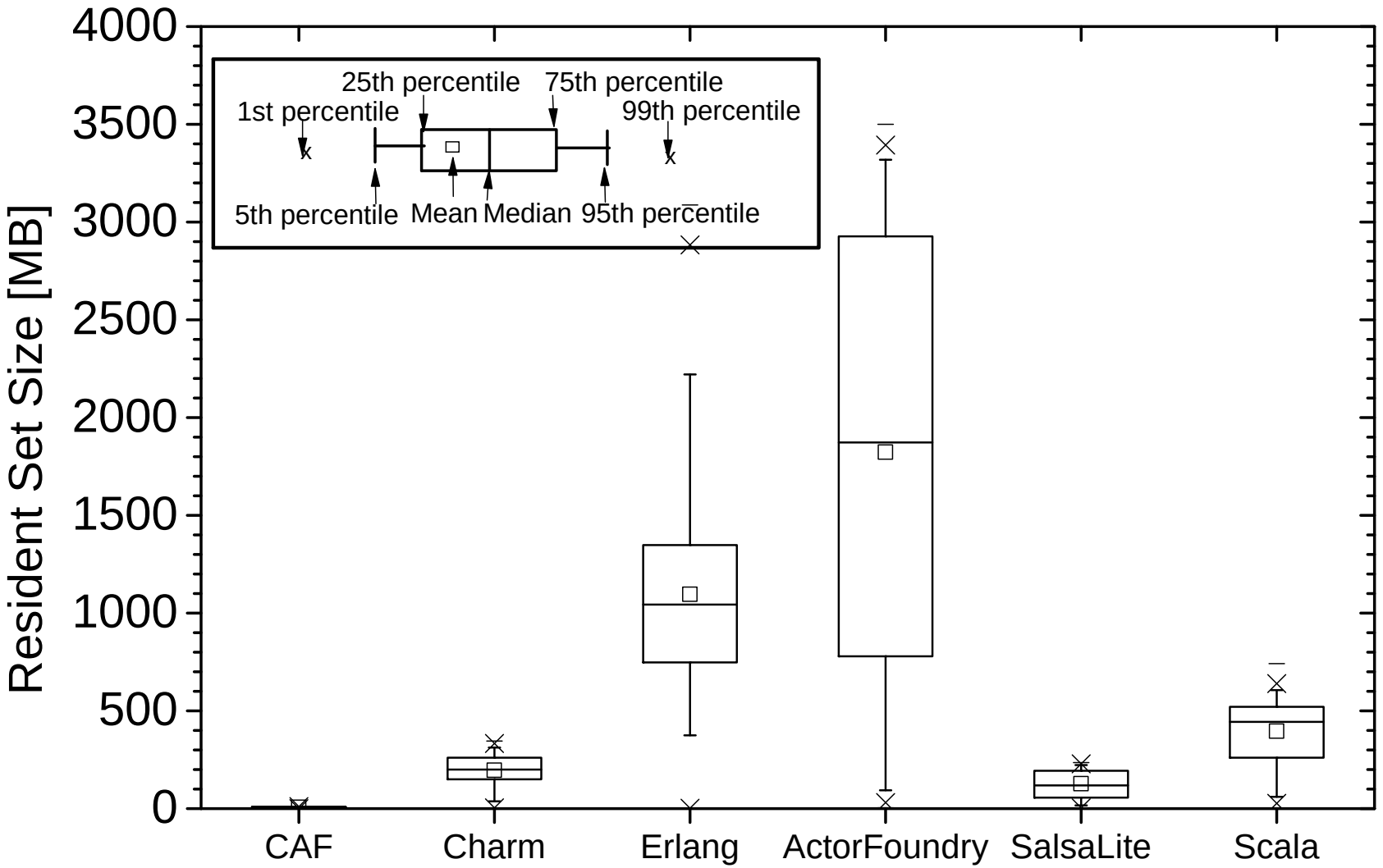
Divide & Conquer

Laufzeit



Divide & Conquer

Speicherverbrauch

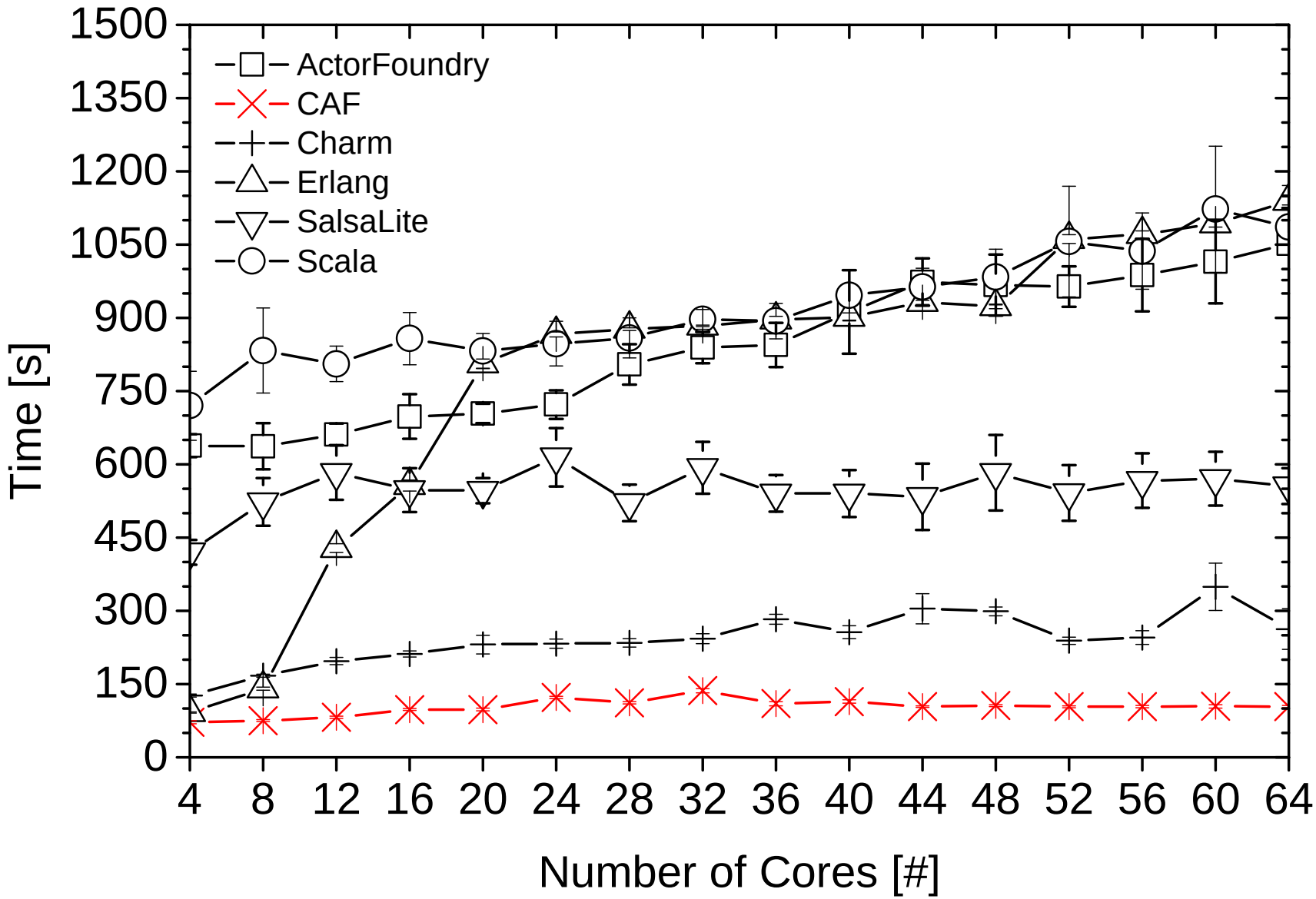


n zu 1 - Kommunikation

- Wie verhält sich die Mailbox unter Last?
- Kritische Komponente: Mailbox
- Viele Sender, ein Empfänger
- Mailbox des Empfängers wird zum Synchronisationspunkt
- Erwartung:
  - Anstieg der Laufzeit
  - Anstieg des Speicherverbrauchs

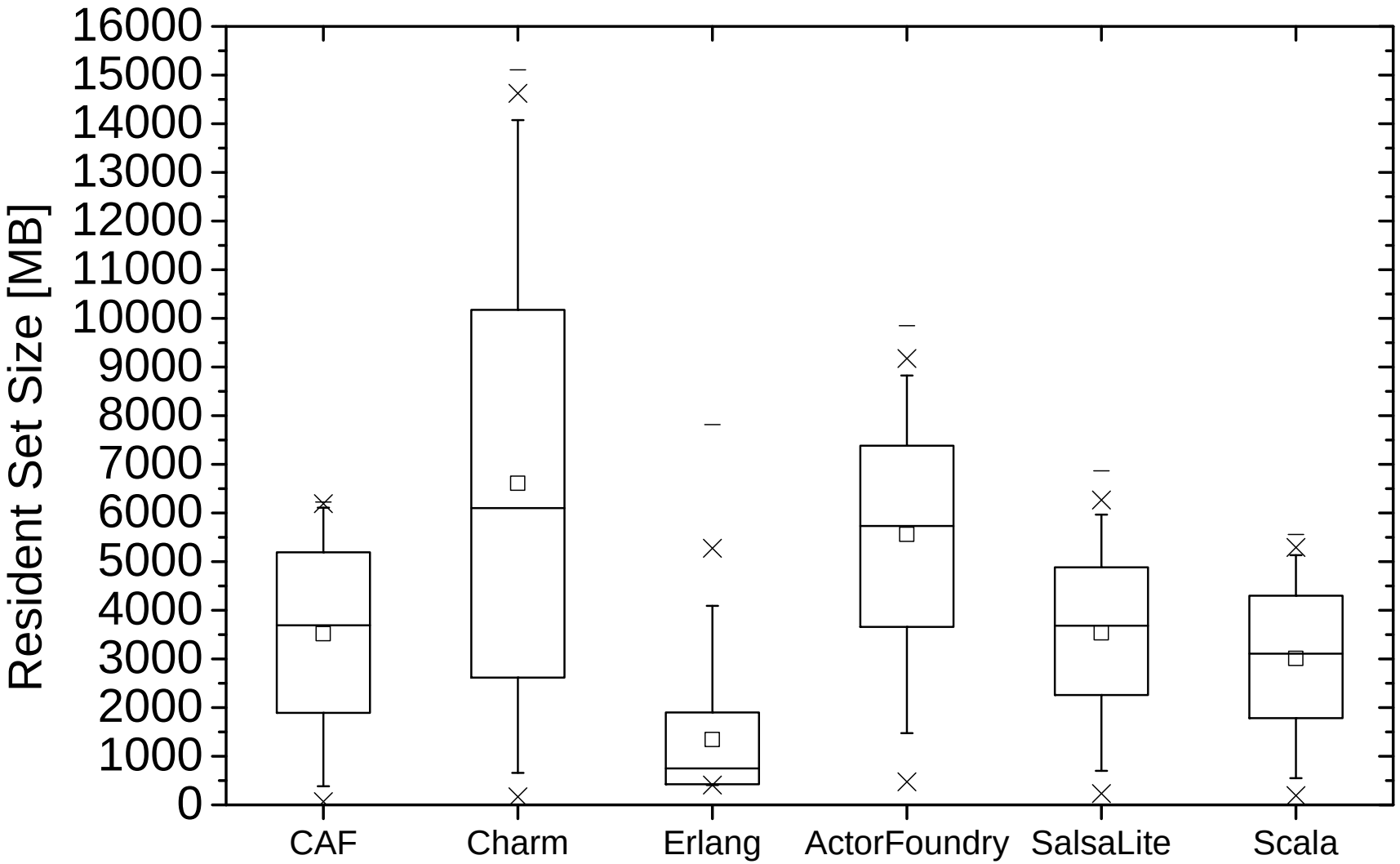
n zu 1 - Kommunikation

Laufzeit



n zu 1 - Kommunikation

Speicherverbrauch

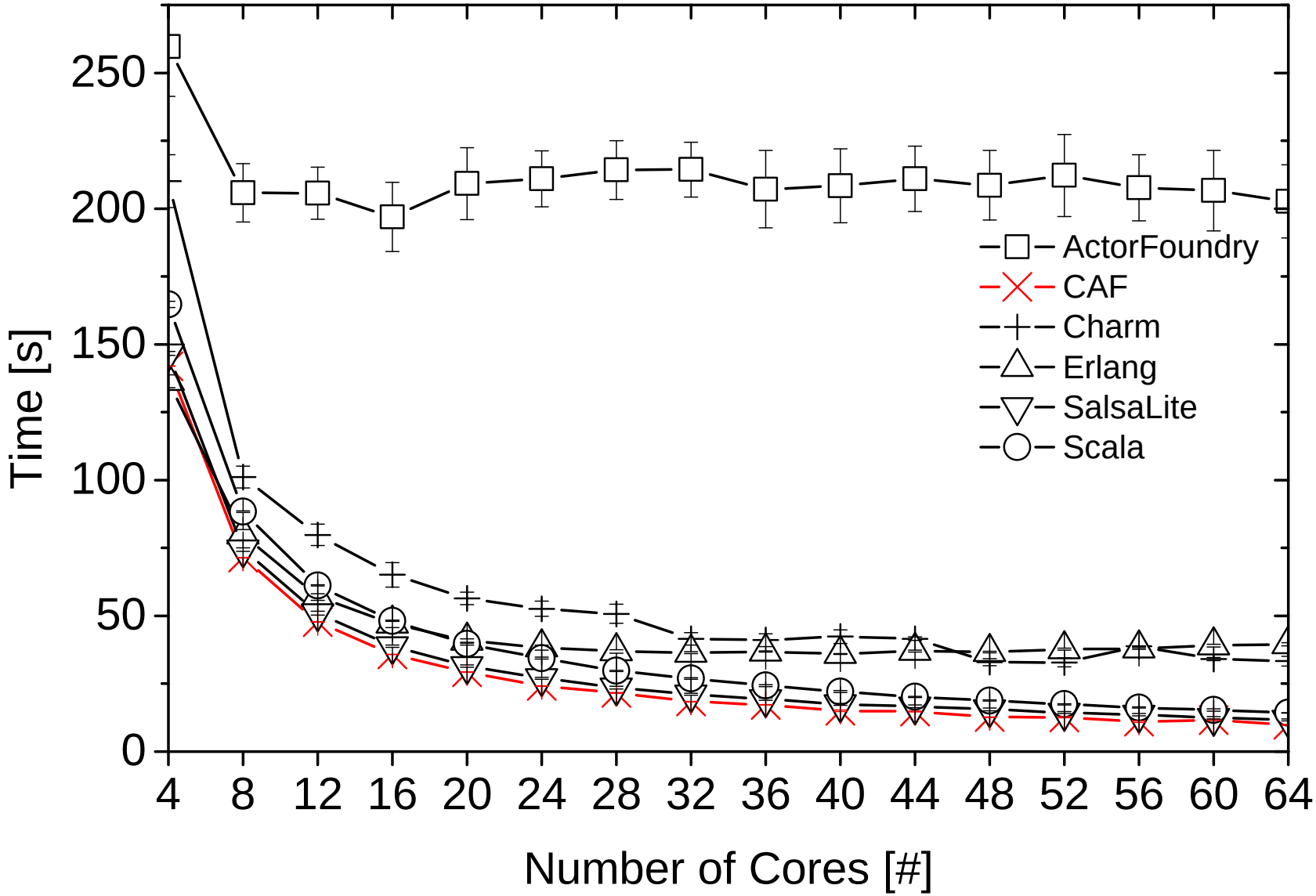


Gemischtes Szenario

- Wie ist das Verhalten bei einem gemischten Szenario?
- Problem: Aktoren werden permanent erzeugt und beendet
- Kommunikation in Ring-Topologie
- Stetiges Neustarten der Ringe
- Zusätzlich Primfaktor-Zerlegung
- Erwartung:
  - Verringerung der Laufzeit
  - Konstanter Speicherverbrauch

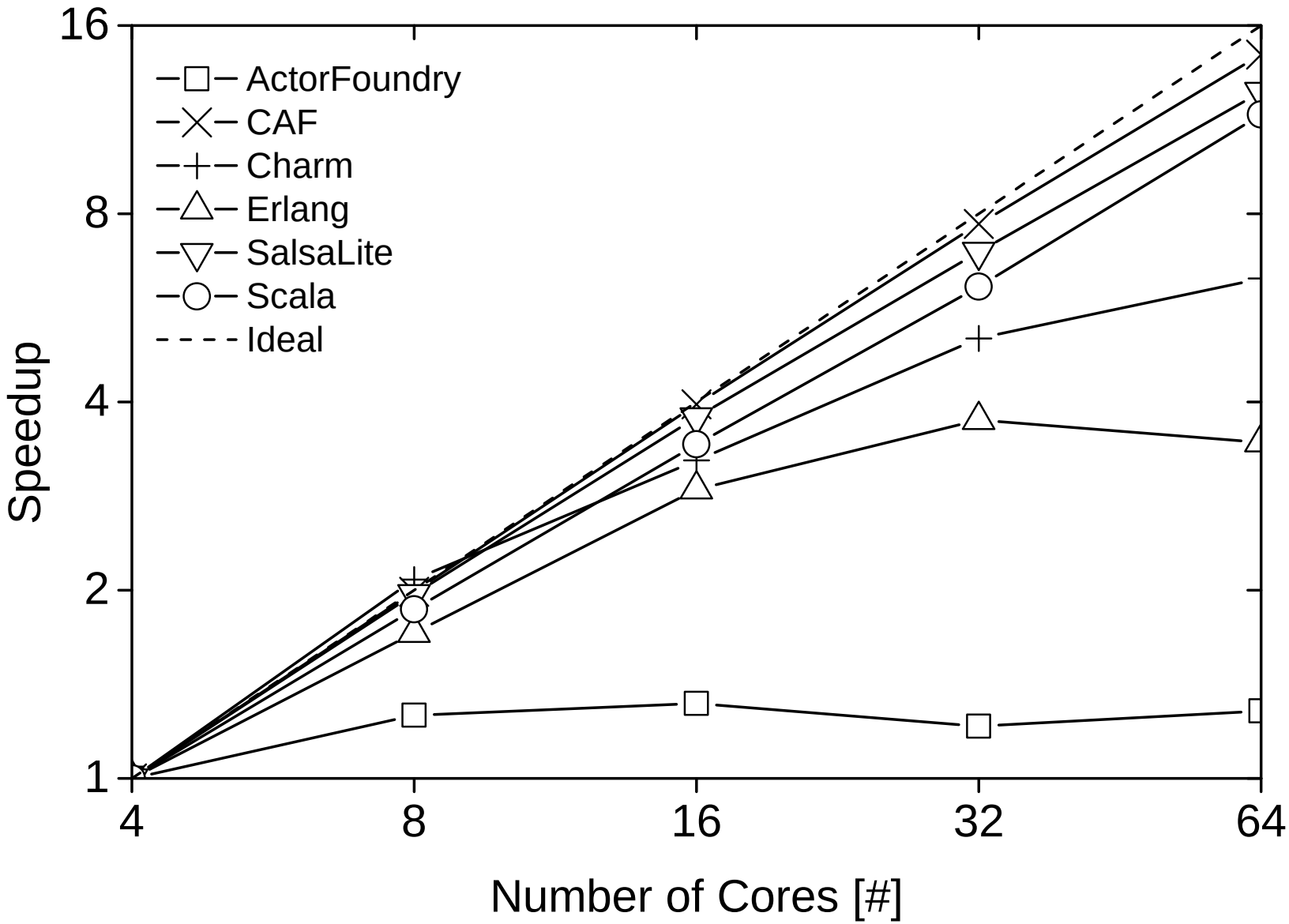
Gemischter Use-Case

Laufzeit



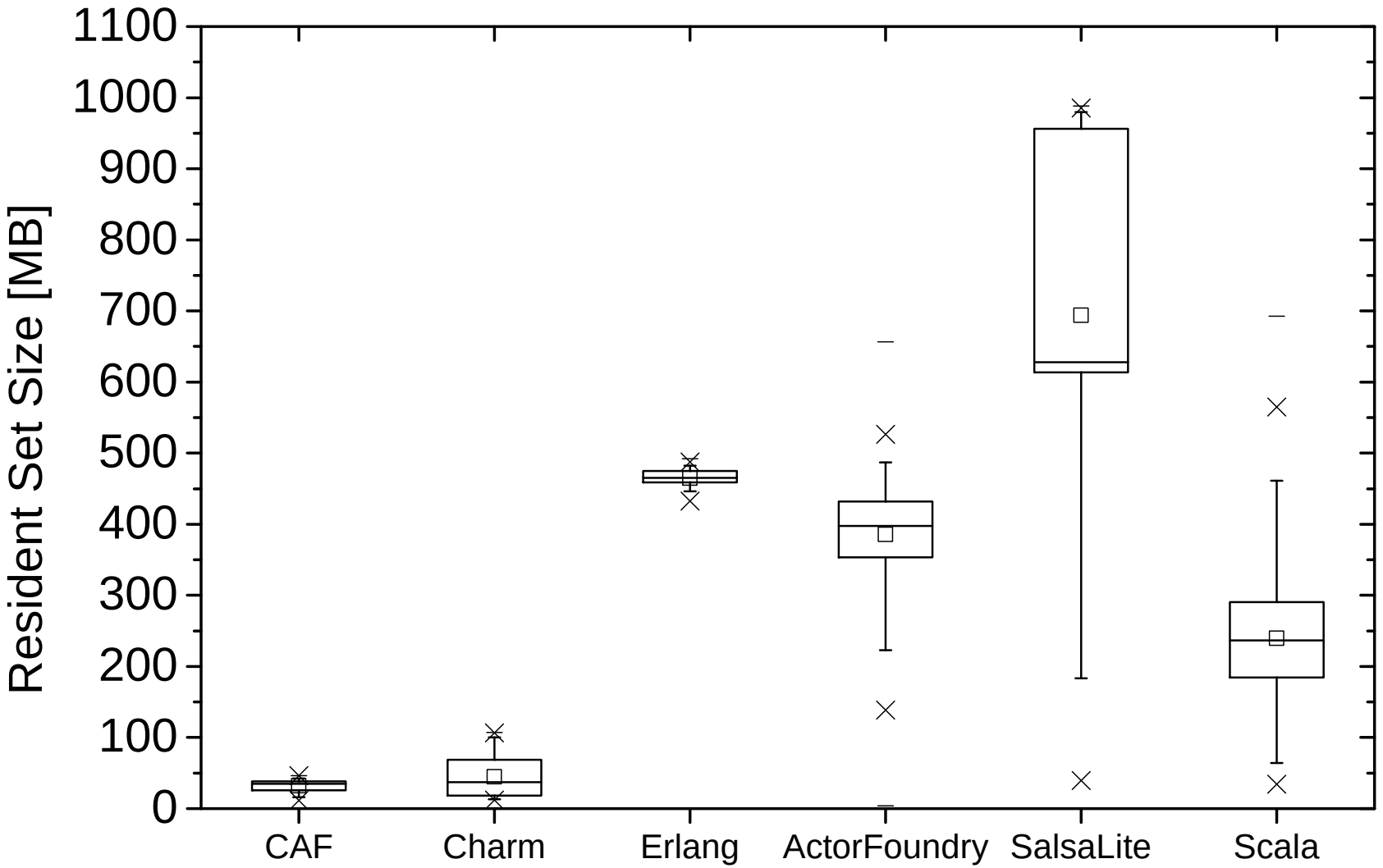
Gemischter Use-Case

Laufzeit



Gemischter Use-Case

Speicherverbrauch



Szenario 2

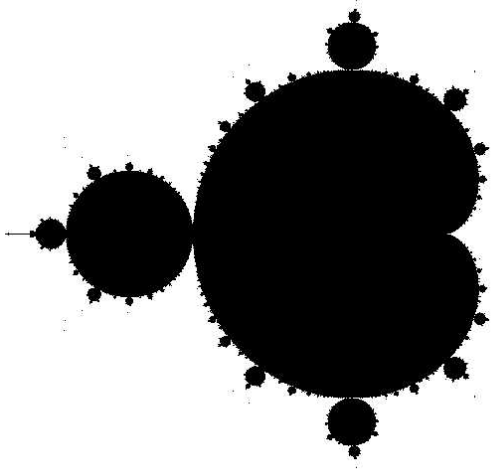
- Verifikation von CAFs Netzwerkabstraktion
- Vergleich von High-Level (CAF) vs. Low-Level (MPI)

Testaufbau 2

- Virtualisierte Systemumgebung
  - Message-Passing
  - Keine Messung der Netzwerkeigenschaften
- Hostsystem:
  - Gleiches System wie zuvor
  - VMs (QEmu)
  - VLAN (OpenVSwitch)

Verteiltes Messagepassing

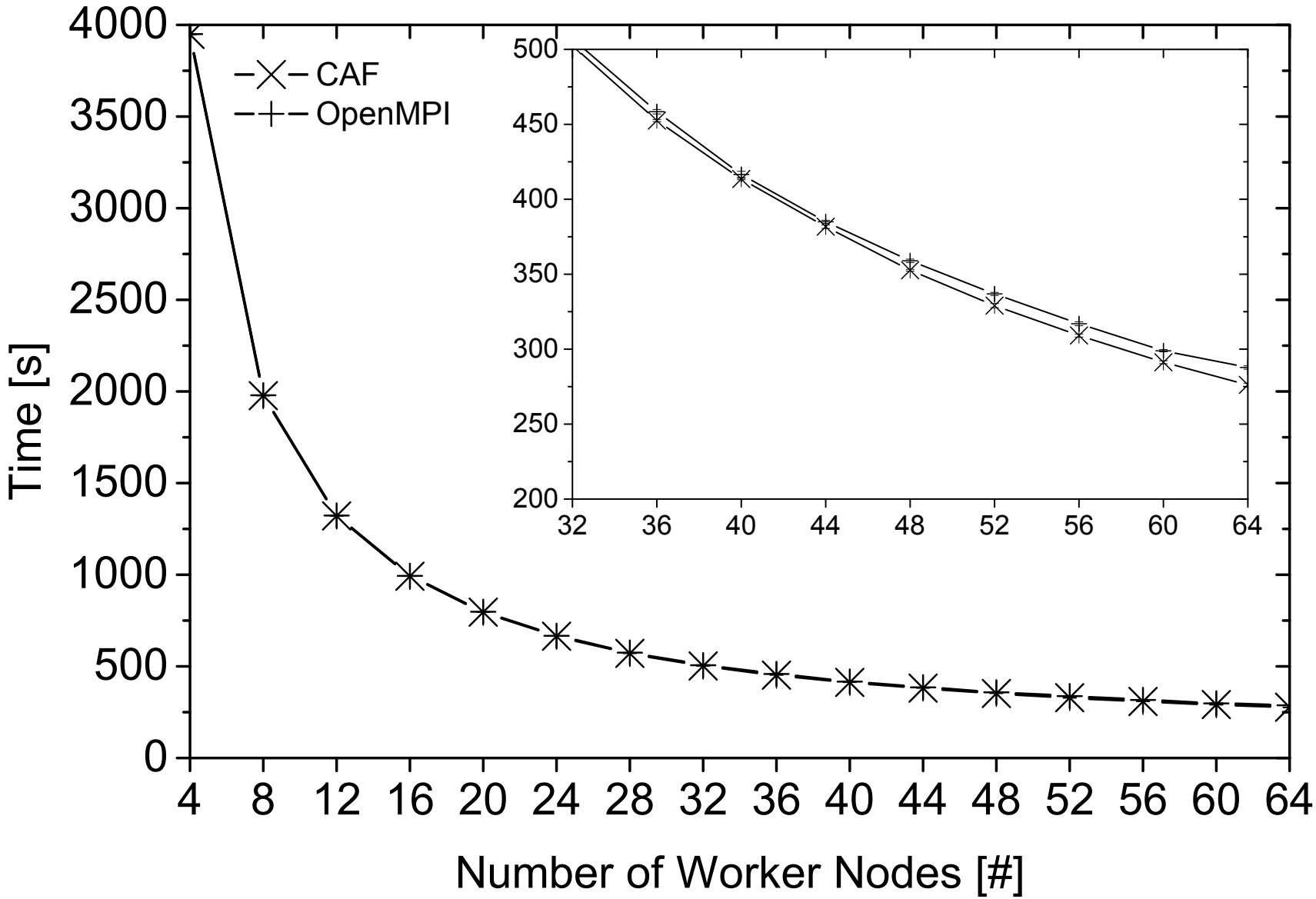
- Führt die höhere Abstraktion in CAF zu Laufzeit-Einbußen?
- Berechnung eines Mandelbrot-Fraktals



- Gleicher Code für Berechnung (C++)
- Erwartung: Halbierung der Laufzeit bei Verdopplung der VMs

Verteiltes Messagepassing

Laufzeit



Szenario 3

- Transparentes Einbinden von heterogener Hardware (hier GPU)
- Messung von eventuellem Laufzeit-Verlust in Zusammenhang mit verschiedenen Problemgrößen

Testaufbau 3

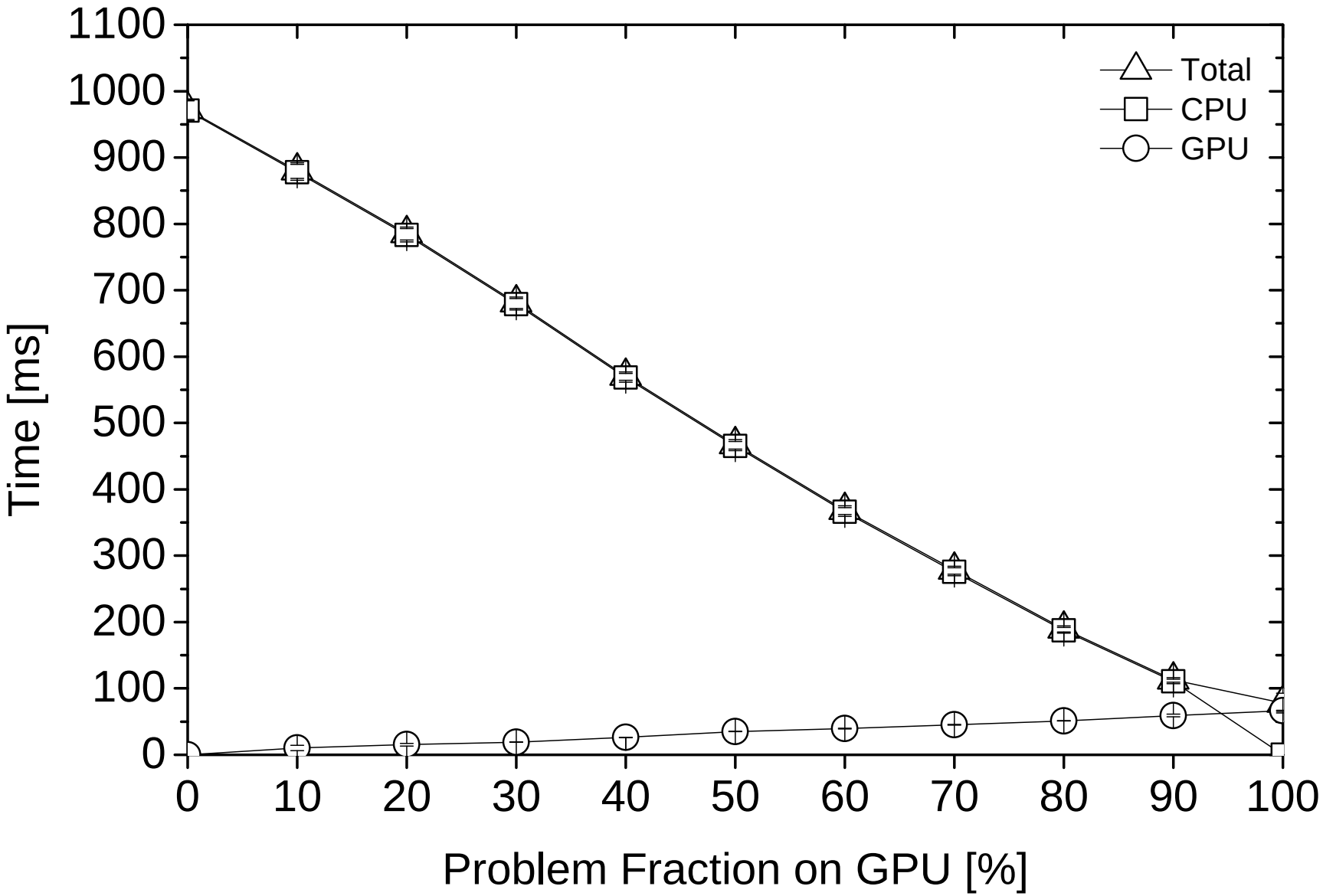
- Hostsystem:
  - Linux
  - 12 Kern Intel Xeon
  - Nvidia Tesla C2075

GPGPU-Wrapper

- Ab wann lohnt es sich eine GPU zu integrieren?
- Kritische Komponente: GPGPU-Wrapper
- Problem wird schrittweise auf GPU verlagert
- Zwei Problemgrößen
  - Mandelbrot kann beliebig detailreich skaliert werden
  - Faktor 10 Detail $\rightarrow$ Faktor 10 Rechenzeit
- Erwartung: Linearer Abfall der Laufzeit

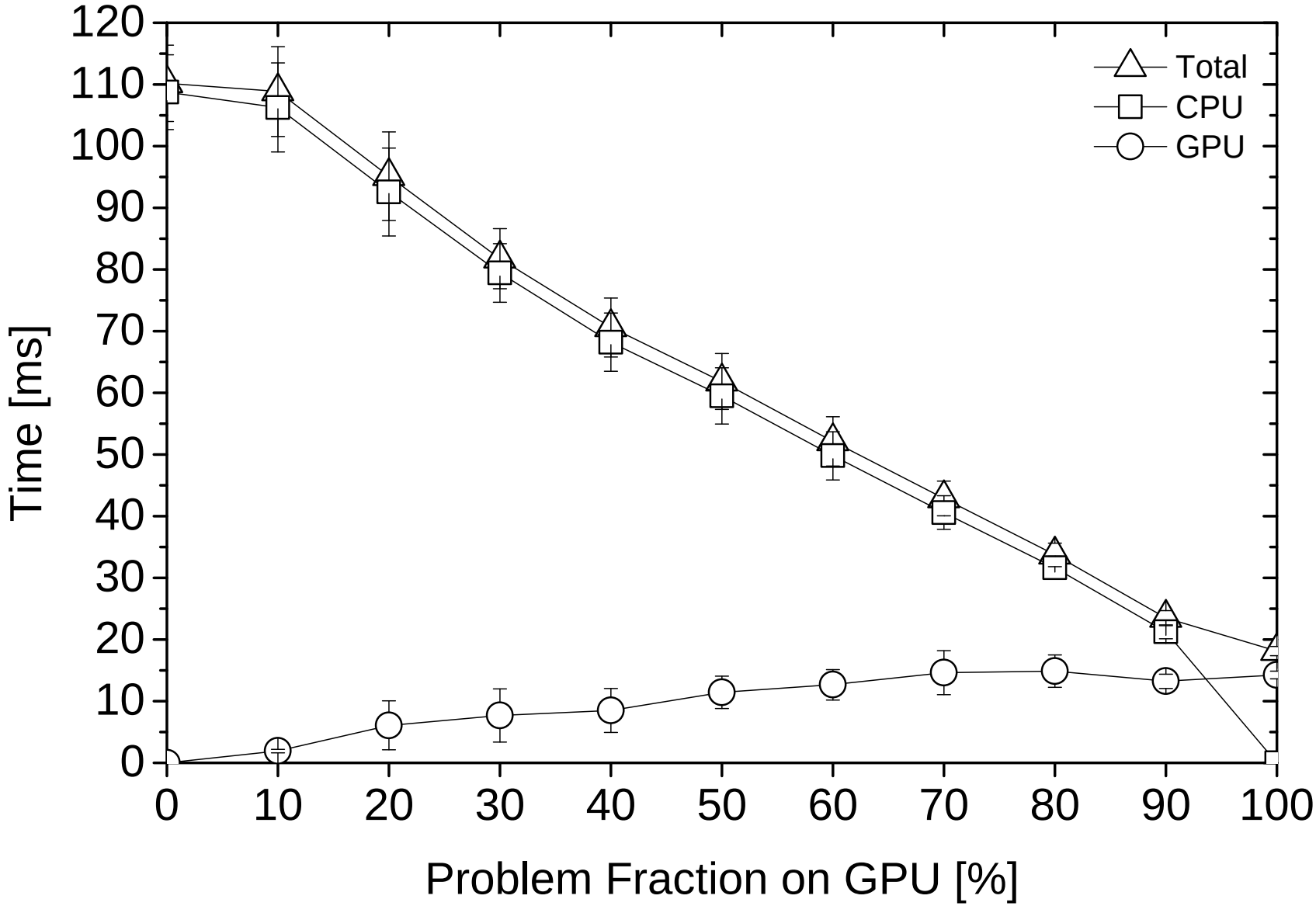
GPGPU-Wrapper

Große Problemgröße



GPGPU-Wrapper

Kleine Problemgröße



Agenda

Fazit

- Geringer Speicherverbrauch
- Durchgehend bestes CPU-Profil sowie kleinen Memory-Footprint
  - Effizientes Scheduling auf Mehrkernsystemen
  - Hohe Performance bei nebenläufigen Mailbox-Zugriff
  - Annähernd ideales Skalierungsverhalten trotz hochstehender Abstraktion
  - Kaum Overhead bei Verwendung des GPGPU-Wrappers

Ausblick

- Scaling Down
  - Internet of Things (IoT)
  - RIOT
- Scaling Out
  - Automatische Lastverteilung
  - Verteiltes Debugging
  - Visualisierungstools

Danke für Ihre Aufmerksamkeit. Fragen?

CAF:	`http://actor-framework.org`
GitHub:	`http://github.com/actor-framework`
iNET:	`http://inet.cpt.haw-hamburg.de`

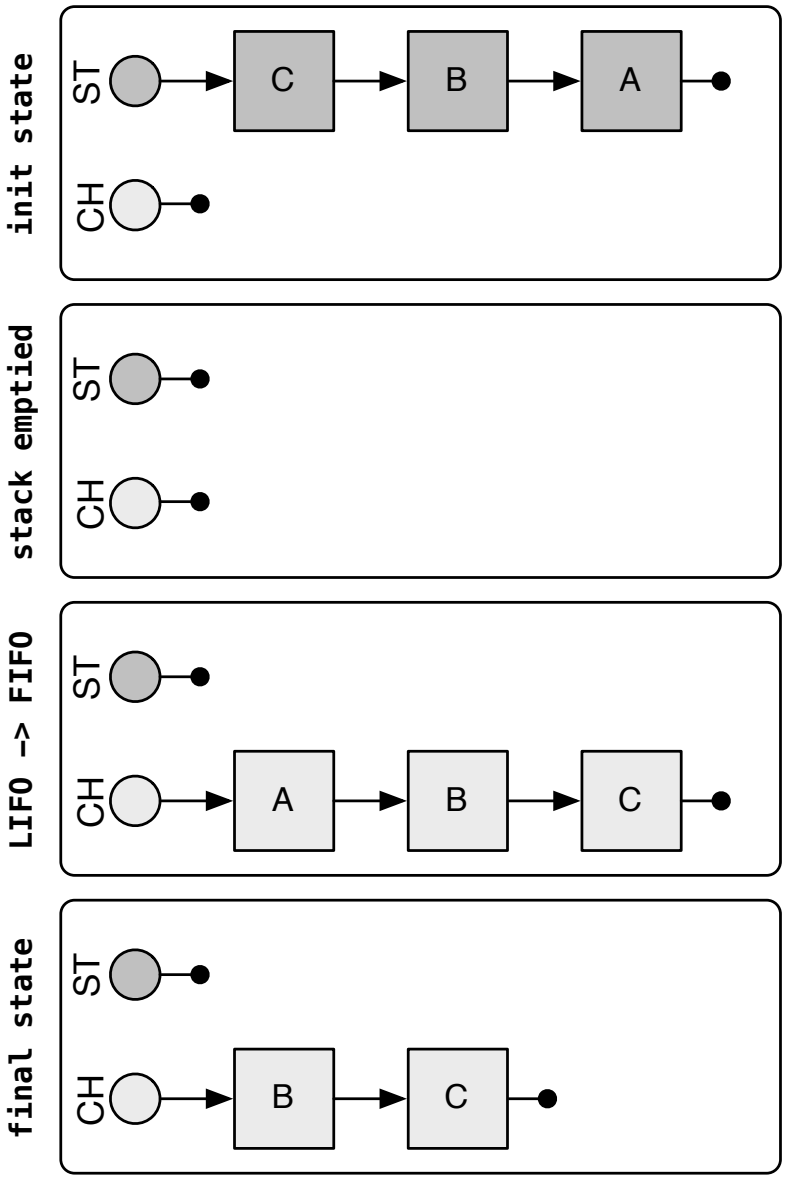
Backup Slides

Nachrichten

- Nachrichten sind Tupel
- Keine Race-Conditions durch Copy-On-Write (COW)
- Aktoren teilen sich Nachrichten
- Eingebauter Referenzzähler (Intrusive)

Backup Slides

Mailbox




```
dequeue()
{
  R = CH
  if R != NULL {
    CH = R.next
    return R
  }


  do {
    E = ST
    if E == NULL
      return NULL
  } while not cas
          (&ST,E,NULL)


  while E != NULL {
    NEXT = E.next
    E.next = CH
    CH = E
    E = NEXT
  }



  return dequeue()


}
```

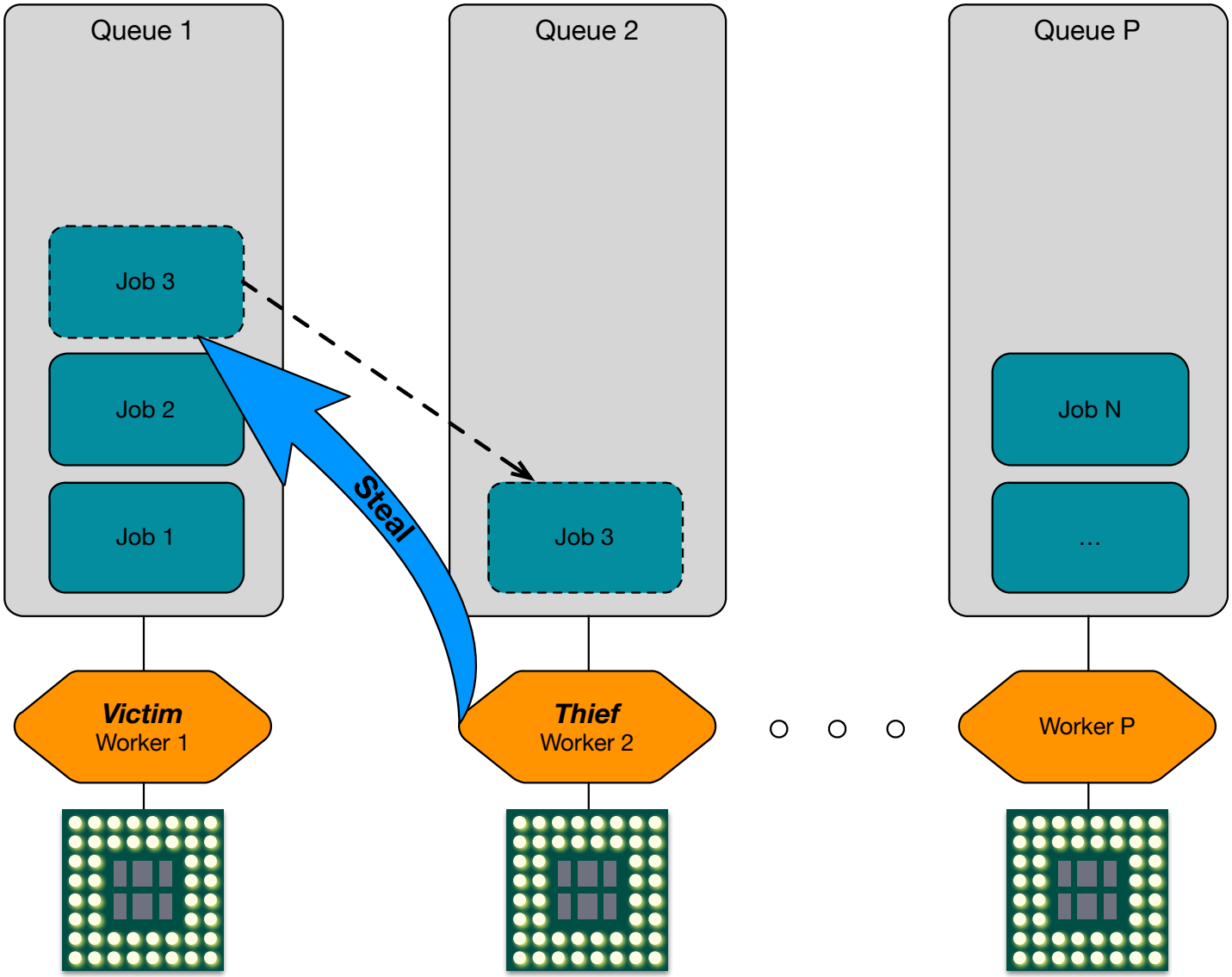
Backup Slides

Work-Sharing & Work-Stealing

- Work-Sharing:
  - Zentrale Queue
  - Lock-/ Signal-basiert
  - Gleichberechtigter Zugriff auf Queue
- Work-Stealing:
  - Jeder Worker hat eigene Queue
  - Polling-basiert
  - Wenn eigene Queue leer: stehlen
  - Bestohlener wird per Zufall gewählt

Backup Slides

Work-Stealing



Backup Slides

Typisierte Aktoren

- Problem: Ungewollt fehlerhafte Signatur in Nachricht
  - Aktoren verwerfen Nachricht
  - Nur die Möglichkeit eines Laufzeitfehlers
  - Besser: Fehler zur Kompilierzeit

Backup Slides

Typisierte Aktoren

```
using math_t = typed_actor<
  replies_to<int,int>::with<int>>;

math_t::behavior_type math_server() {
  return {
    [](int a, int b) { return a + b; }
  };
}
void math_client(event_based_actor* self, math_t ms) {
  self->sync_send(ms, 40, 2).then(
    [=](int result) {
      cout << "40 + 2 = " << result << endl;
    }
  );
}
// spawn(math_client, spawn_typed(math_server));
```