# Organisatorisches und Überblick

Prof. Dr. David Sabel

LFE Theoretische Informatik


Letzte Änderung der Folien: 2. November 2020

# Beteiligte und Ressourcen

**Prof. Dr. David Sabel**

- Raum L107, Oettingenstr. 67
- david.sabel@ifi.lmu.de

**Links zur Veranstaltung:**

- Webseite: `https://www.tcs.ifi.lmu.de/lehre/ws-2020-21/concurrent/`
- Im uni2work: `https://uni2work.ifi.lmu.de/course/W20/IfI/PAMNP`
- Zulip-Stream zur Veranstaltung: `TCS-20W-PAMNP-VL` via `chat.ifi.lmu.de`
  Fragen am besten dort stellen!

**Material** (Verfügbar nach Anmeldung in uni2work):

- Aktuelle und organisatorische Informationen, Zoom-Link für die Übung!
- Unterlagen zur Veranstaltung: Skript, Folien, Aufgaben, Videos!
- Referenzen auf Bücher, Webseiten, Programmiersprachen usw.

# Termine

**Vorlesung**

- Mittwoch, 10-12
- Freitag, 10-12 ca. alle 2 Wochen

  Vorläufiger Plan für die Freitage:

  06.11.20 20.11.20 04.12.20 18.12.20 15.01.21 29.01.21 12.02.21

- Die Vorlesung findet **asynchron** statt (Aufzeichnung als Screencast)
- Im uni2work unter Material als **Virtuelle Vorlesung**:
  Für jeden Vorlesungtermin: Angabe des passenden Material (Video, Skript, Folien)

**Übung**

- Dienstags 16-18, **live** via Zoom
- Fragen zur Vorlesung und Besprechung von Aufgaben

# Aufgabenblätter

- Ungefähr: Jede Woche ein Aufgabenblatt
- Abgabe Dienstags bis 16 Uhr
- 1. Aufgabenblatt wird nicht abgegeben / korrigiert aber nächste Woche besprochen
- Abgabe und Korrektur über uni2work
- Eine Aufgabe pro Blatt wird bepunktet
- 2 Punkte pro Aufgabe
- Bonus für die Prüfung (nächste Folie)

# Hörerkreis und Prüfung

- Modul Algorithmik und Komplexität im Masterstudiengang Informatik
- Anrechenbar als Vertiefendes Fach im Bachelor Informatik
- 6 ECTS, 3V+2Ü

**Modulprüfung**

- Planung: 90-minütige Präsenzklausur (beantragt, noch nicht genehmigt)
- „Bonus“ bei erfolgreicher Bearbeitung der Übungsaufgaben:
  - Alle Übungspunkte = 10% aller Klausurpunkte als Bonus
  - Anrechnung ansonsten linear
  - Bonuspunkte helfen nicht beim Bestehen

# Inhalt (Planung) (1)

**Einleitung**

- Begriffe, Notationen, Annahmen
- Nebenläufige Programme in Java (sehr kurz, für die Beispiele)

**Synchronisation** von Prozessen und **Mutual-Exclusion**

- Mutual-Exclusion = wechselseitiger Ausschluss
- Unter verschiedenen Annahmen der Speicheroperationen:
- Zunächst: Nur atomares read und write
- Mutual-Exclusion bei 2 Prozessen:
  Algorithmen von Dekker, Peterson, Kessels
- Mutual-Exclusion bei $n$ Prozessen:
  Lamports Algorithmus, Bakery-Algorithmus
- Komplexitätsresultate: Wieviele Speicherobjekte braucht man mindestens bzw. höchstens? Wie lange müssen Prozesse warten?

# Inhalt (Planung) (2)

**Stärkere Speicherobjekte**

- Stärke Speicheroperationen und Mutual-Exclusion: Test-and-set Bits, RMW-Objekte, Compare-and-Swap-Objekte, ...
- Qualitative Klassifizierung der Speicheroperationen
- Konsensus und die Konsensuszahl

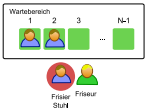
# Inhalt (Planung) (3)

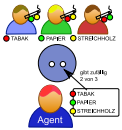
**Programmierprimitive zur Nebenläufigen Programmierung**

- Semaphore
- Monitore
- Kanäle
- Tuple Spaces

Dabei: Lösung klassischer Probleme mit den Primitiven:

- Das Problem der Speisenden Philosophen
- Erzeuger / Verbraucher Probleme
- Sleeping Barbers Problem
- Cigarette Smoker's Problem
- Programmierung von Barrieren
- Das Readers & Writers Problem



# Inhalt (Planung) (4)

**Nebenläufiger Zugriff auf mehrere Ressourcen**

- Deadlocks
- Deadlock-Vermeidung (Zwei-Phasen-Sperrprotokoll)
- Deadlock-Verhinderung (Bankiers Algorithmus)
- Transactional Memory (allgemein)

# Inhalt (Planung) (5)

**Nebenläufige Programmierung in Haskell**

- Kurze Einführung in Haskell
- Concurrent Haskell
- STM Haskell

Andere Programmiersprachen werden innerhalb der vorherigen Kapitel auch kurz behandelt: Geplant Java und Go

# Inhalt (Planung) (6)

Semantische Modelle nebenläufiger Programmiersprachen:

- Kurze Einführung in die Semantik von Programmiersprachen
- der $\pi$-Kalkül als Message-Passing-Modell
- der CHF-Kalkül als Shared-Memory-Modell
- Beziehungen zwischen den Modellen
- evtl. weitere Modelle

# Literatur (1)


Zu Mutual-Exclusion-Algorithmen und Konstrukte der nebenläufigen Programmierung

**Ben-Ari, M.** (**2006**). *Principles of concurrent and distributed programming*. Addison-Wesley.

**Taubenfeld, G.** (**2006**). *Synchronization Algorithms and Concurrent Programming*. Prentice-Hall.

**Reppy, J. H.** (**2007**). *Concurrent Programming in ML*. Cambridge University Press.

**Herlihy, M. und Shavit, N.** (**2008**). *The Art of Multiprocessor Programming*. Morgan Kaufmann Publishers Inc.

**Raynal, M.** (**2013**). *Concurrent Programming: Algorithms, Principles, and Foundations*, Springer.

# Literatur (2)

Zu Concurrent Haskell:

- **Peyton Jones, S. & Singh, S. (2009)**. A tutorial on parallel and concurrent programming in Haskell. In *Advanced Functional Programming, 6th International School, AFP 2008, Revised Lectures*, LNCS 5832, S. 267–305. Springer.
- **Marlow, S. (2013)**. *Parallel and Concurrent Programming in Haskell*, O'Reilly.

...

Zum $\pi$-Kalkül:

**Milner, R. (1999)**. *Communicating and Mobile Systems: the Pi-Calculus*. Cambridge University Press.

**Sangiorgi, D. & Walker, D. (2001)**. *The $\pi$-Calculus: A Theory of Mobile Processes*. Cambridge University Press, New York, NY, USA.

Verweise auf speziellere Forschungsartikel etc. werden im Skript zu den einzelnen Themen angegeben.