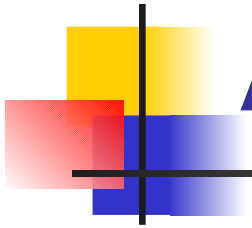
University of Karlsruhe
Institute of Computer Science and Engineering

Yinspire – Ein performance-effizienter Simulator für gepulste Neuronale Netze

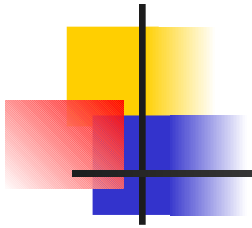
Michael Neumann

Studienarbeit, 2008

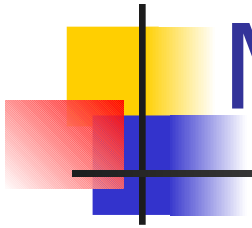


Agenda

- Motivation
- Simulator-Modelle
- Yinspire
- Fazit



Motivation

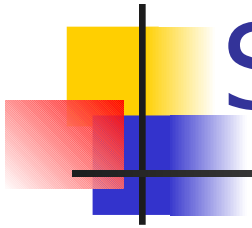


Motivation

- Simulation immer komplexer werdender Netze
- Simulation vieler Netze
  - z.B. Parameterraum-Untersuchungen mittels Evolutionärer Algorithmen
- Implementierung neuer Modelle

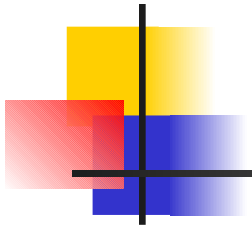
Primäres Ziel

- Hohe Performanz

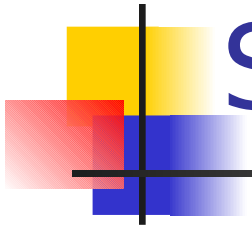


Sekundäres Ziel

- Erweiterbarkeit
- Wartbarkeit
  - Lesbarkeit des Quelltextes



Simulator-Modelle



Simulations-Modelle

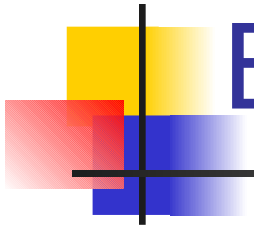
- Zwei verschiedene Ansätze:
  - Zeitschritt Verfahren
  - Ereignis-gesteuerte Simulation

Zeitschritt-Verfahren

- Führe Zustand des *kompletten* Netzes zu Zeitpunk t_i in Zustand $t_i+\Delta t$ über.
- Ähnlich:
  - Zellulare Automaten mit beliebiger Nachbarschaft.
  - Finite Elemente

Zeitschritt-Verfahren

- Pro
  - Hochgradig parallelisierbar
  - Einfach zu implementieren
  - Konstanter Zeitaufwand, d.h. unabhängig von Aktivität im Netz
- Contra
  - Quantisierung der Zeit
  - Trade-Off:
    - Geschwindigkeit - Genauigkeit

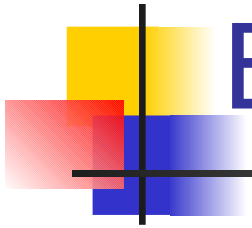


Ereignis-gesteuert

- Betrachte Ereignisse die zu pot. Zustandsänderung führen
  - Ereignis = “Feuern eines Neurons”
    - Wann feuert es (Zeitpunkt)
    - Wie stark
  - Neuberechnen von Ereignis betroffener Elemente
- Verwalten der Ereignisse ist Hauptaufgabe des Simulators

Ereignis-gesteuerter Ablauf

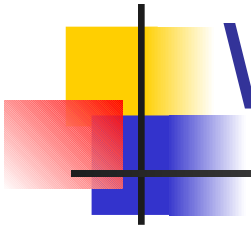
1. *event = removeMin()*
   - Zeitlich nächstes Ereignis holen
2. *event.target.process(event.at)*
   - Neuberechnen des betroffenen Elements
     - Kann zu neuen Ereignissen führen: *insert(newEvent)*
3. Gehe zu 1.



Ereignis-gesteuert

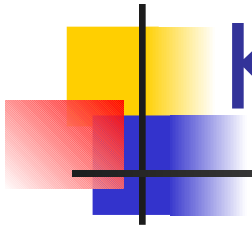
- Pro:
  - Effizient bei geringer Aktivität des Netzes
  - “Zeitwarps”
  - Exakte Zeit (keine Quantisierung)
- Contra:
  - Schlecht parallelisierbar
  - Trade-Off: Geschwindigkeit - Aktivität
  - Nur für spez. Neuronen Modelle.

Yinspire



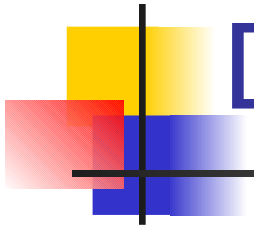
Was ist Yinspire

- Ereignis- und Prozess-orientierter Simulator für gepulste Neuronale Netze
- Nachfolger von Inspire
- Komplett neu geschrieben (in C++)
- Neuartige Architektur
- Interfaces für Matlab und Ruby, neues File-Format...



Kerngedanken

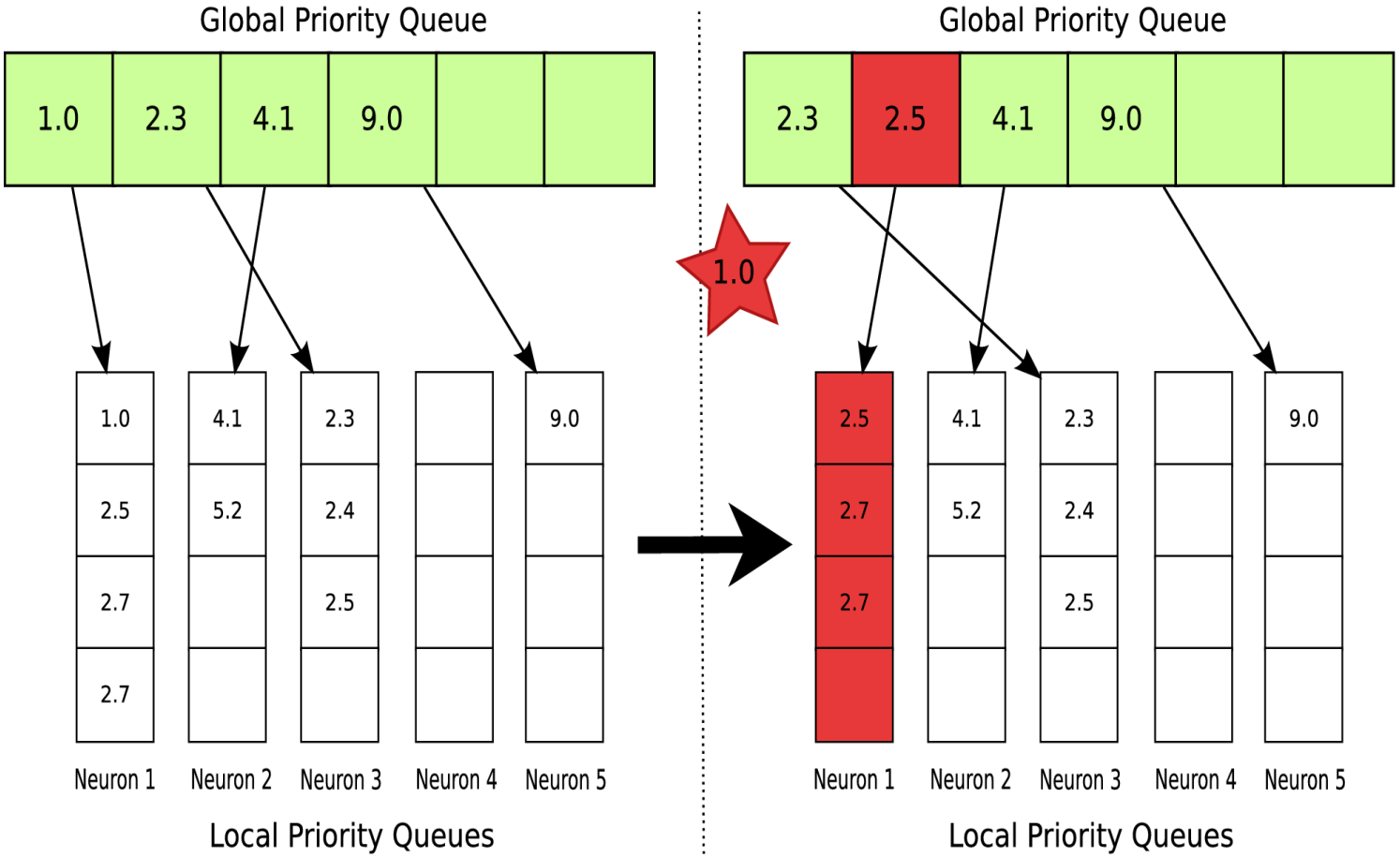
- Dezentralisierung
  - 1 globale + n lokale Warteschlangen
- Kapselung von Verhalten
  - Entkopplung



Denzentralisierung

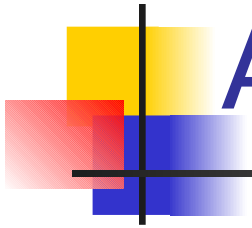
- Jedes Element hat eigene lokale Warteschlange.
  - Enthält alle Ereignisse für dieses Element.
  - Lokalen Warteschlangen sind hierarchisch zu einer globalen Warteschlange zusammengeschaltet.

Dezentralisierte PQ



Dezentralisierung

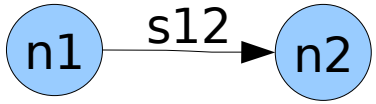
- Gleicht mehr der Natur.
- 30% reduzierter Speicherbedarf.
- Jedes Element ist selbst zuständig für die Speicherung.
  - Beliebige Algorithmen sind denkbar
  - z.B. Aggregation (mgl. adaptiv)
  - z.B. Verzicht auf Speicherung innerhalb der absoluten Refraktären Phase

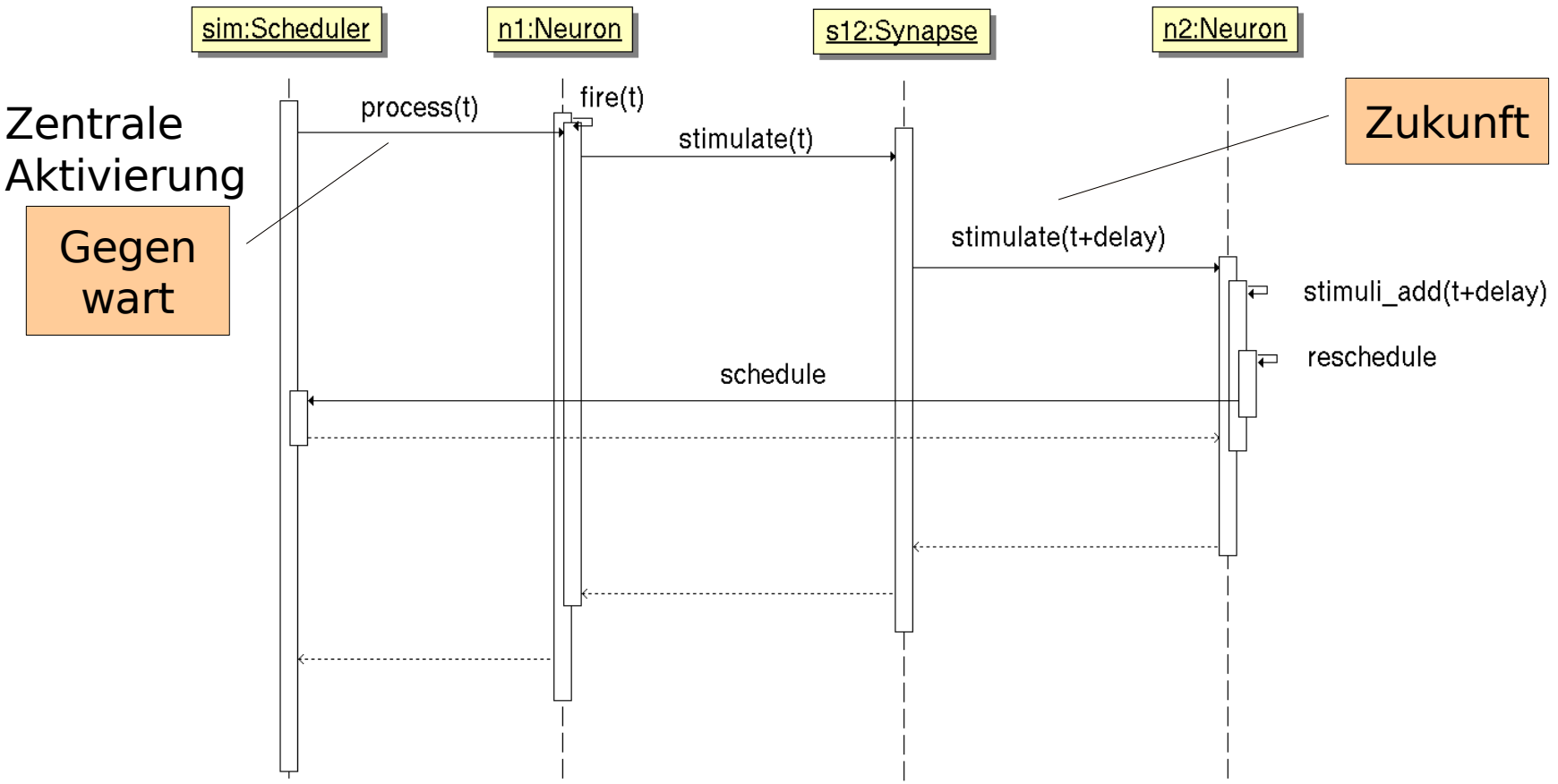


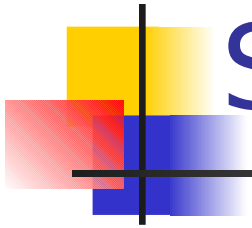
Ablauf

- Zwei Phasen
  - Stimulation
    - z.B. beim “Feuern”
    - Generieren von Ereignissen die in der Zukunft passieren
    - *stimulate*
  - Ereignis-Verarbeitung
    - Verarbeiten des Ereignisses, das in der Gegenwart angelangt ist.
    - *process*

Ablauf


n1
s12
n2


sim:Scheduler
n1:Neuron
s12:Synapse
n2:Neuron
Zentrale Aktivierung
Gegen wart
Zukunft
process(t)
fire(t)
stimulate(t)
stimulate(t+delay)
stimuli_add(t+delay)
reschedule
schedule

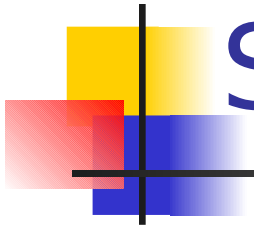


Stimulation

- Elemente “senden” *Stimuli*:
  - Methode *stimulate(at, weight, origin)*
    - Wann?
    - Wie stark?
    - Von wo kommt der Stimulus?

Stimulation (Neuron)

- Beispiel Neuron feuert:
  - $\forall\, s \in postSynapsen:\ s.stimulate(jetzt, \ldots)$



Stimulation (Synapse)

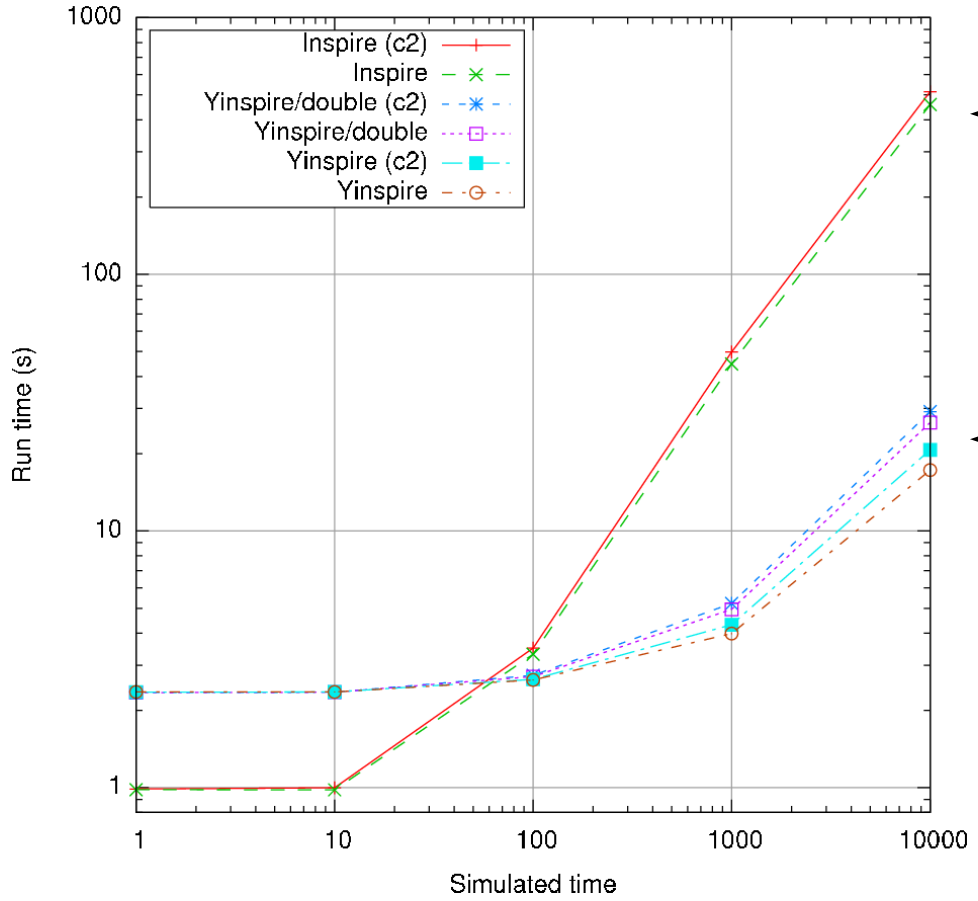
- Beispiel Synapse:
  - Leitet Stimuli an Post-Neuron weiter
  - Verzögert: *stimulate(t+delay)*
  - Mit entsprechendem Gewicht der Synapse

Ereignis-Verarbeitung

- Element kann zu gewünschter Zeit vom Simulator aktiviert werden
  - Entity: *simulator.schedule(at)*
  - Simulator: *entity.process(at)*
- Entfernen aller Ereignisse aus lokaler Warteschlage zur Zeit “at”
- Inneren Zustand neu berechnen
- Evtl. Feuern

Fazit

Schneller

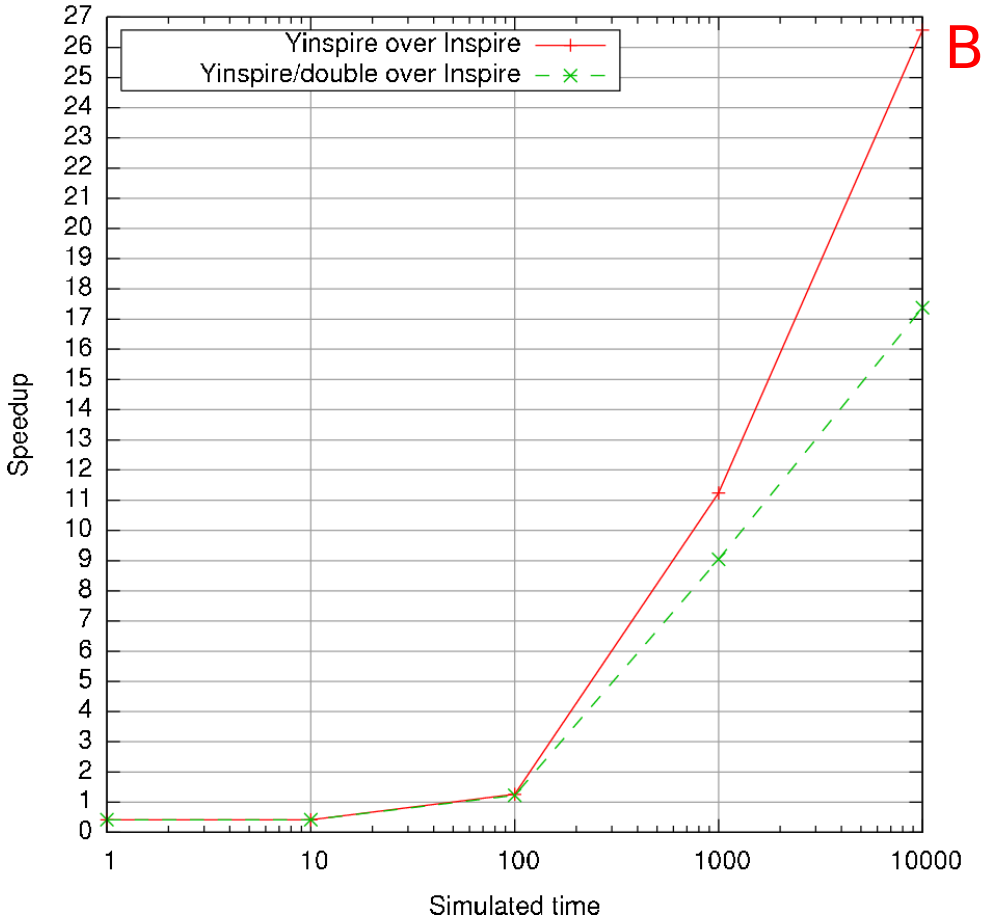


ca. 400 sec.

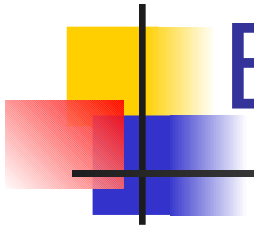
> 20x

ca. 20 sec.

Schneller



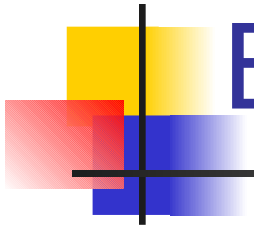
Bis zu 27x schneller!



Einfacher

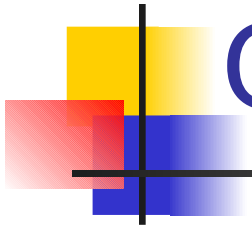
- Inspire (Gereon 2005)
  - ~3000 Zeilen
- Inspire (SVN)
  - ~10.000 Zeilen
    - ~4000 GraphML
    - ~6000 Simulator
- Yinspire
  - ~4000 Zeilen

Alles ohne Leerzeilen und Kommentare!



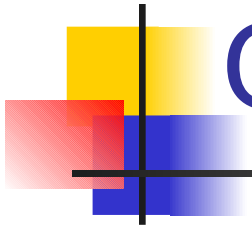
Einfacher

- Neuron_SRM01 (KernelbasedLIF)
  - Inspire: 272 Zeilen (529 inkl. Hebbsch)
  - *Yinspire: 105 Zeilen*
- Neuron_LIF01 (ECurLIF)
  - Inspire: 306 Zeilen
  - *Yinspire: 142 Zeilen*



Genereller

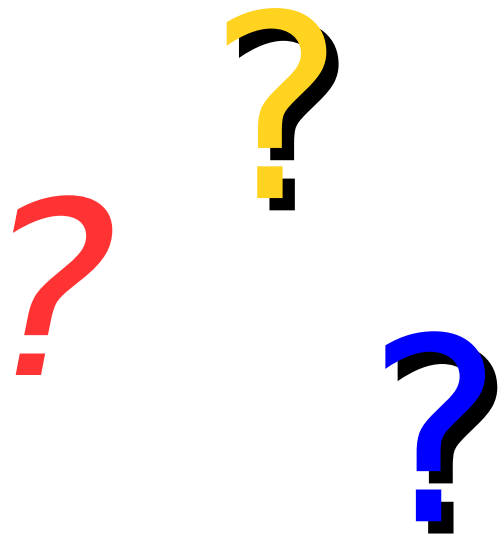
- Kapselung von Verhalten
  - Kein spezielles Wissen über andere Modelle (z.B. Neuronen über Synapsen)
  - Dazu gehört die Speicherung von Ereignissen in lokalen Warteschlangen
    - Ermöglicht z.B. hierarchische Netze

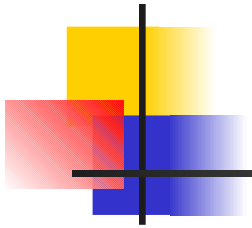


Genereller

- Jedes Neuronen-Modell kann sich “Hebbsch”-Verhalten.
- Vereinheitlichung von Ereignis- und Prozess-orientiertem Ansatz
  - Jedes Entity kann beide Ansätze verwenden.

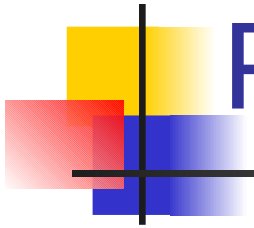
Vielen Dank!





Backup

Prioritäts-Warteschlangen

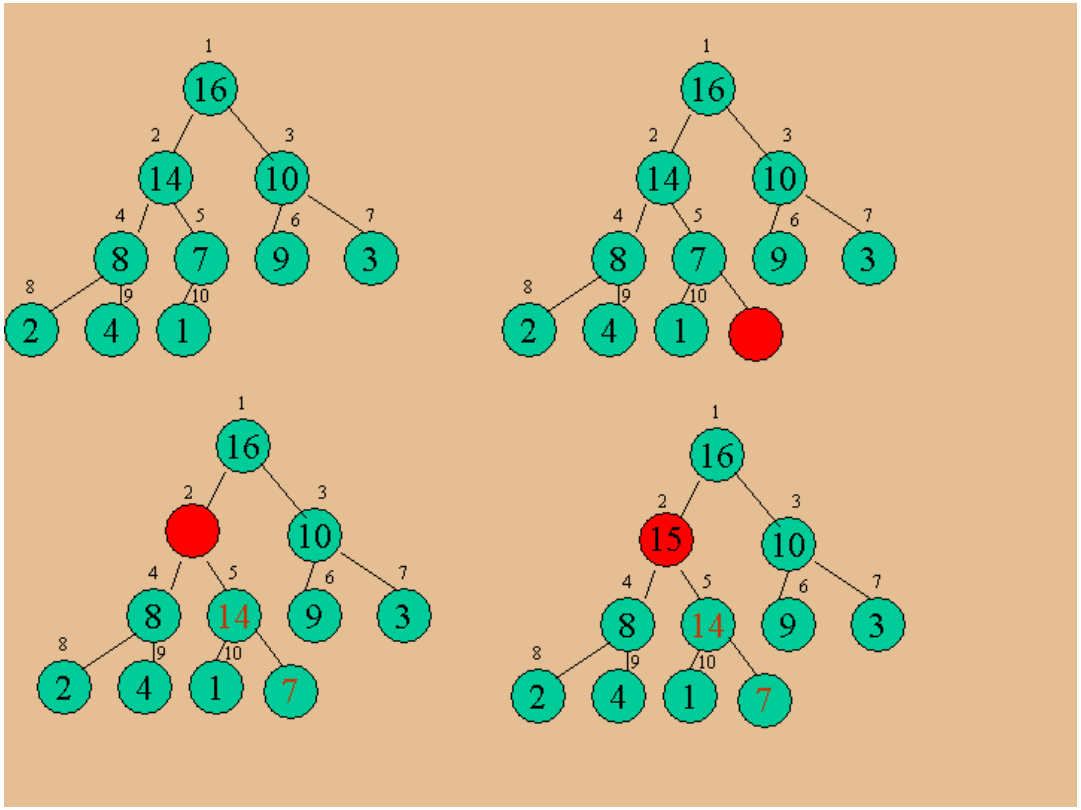


Prioritäts-Warteschlangen

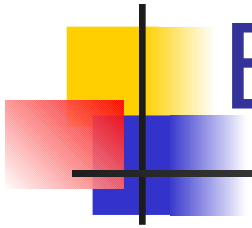
- Zeit als Priorität (“Feuerzeitpunkt”)
- Operationen:
  - extractMin()
    - Liefert Ereignis mit höchster Priorität, d.h. “frühester” Zeit
  - removeMin()
    - Entfernt dieses
  - insert(Ereignis)
    - Hinzufügen von beliebigem Ereignis

Binärer Heap

Einfügen von “15”



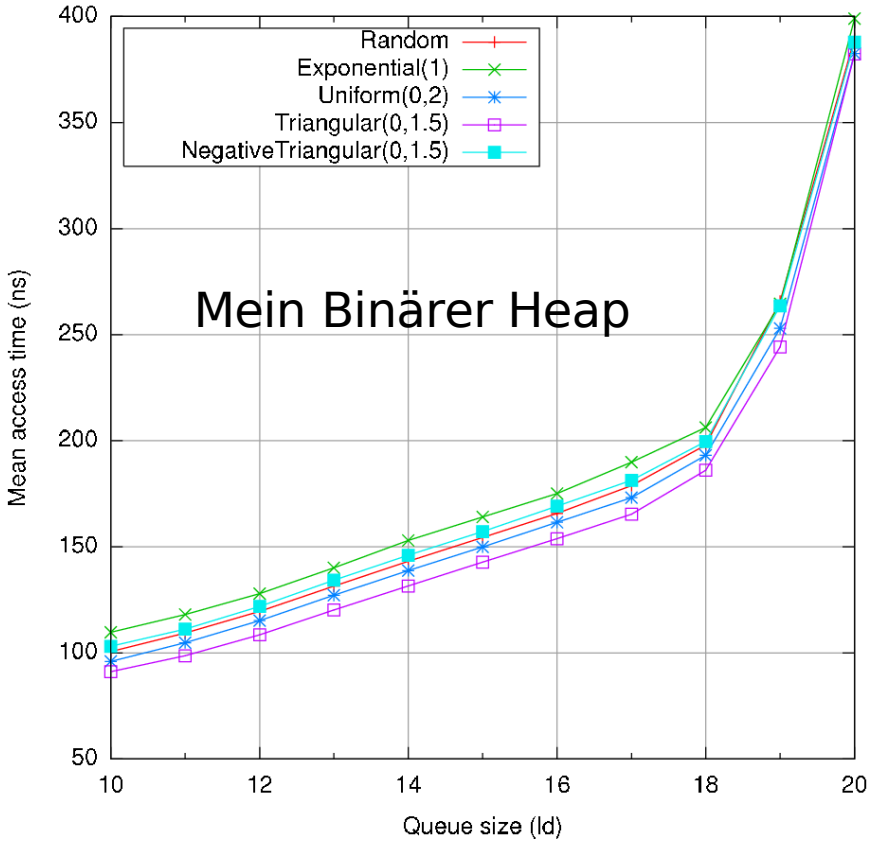
Quelle: http://www.cs.fsu.edu/~cop4531/slideshow/images_content/7-2-7.gif

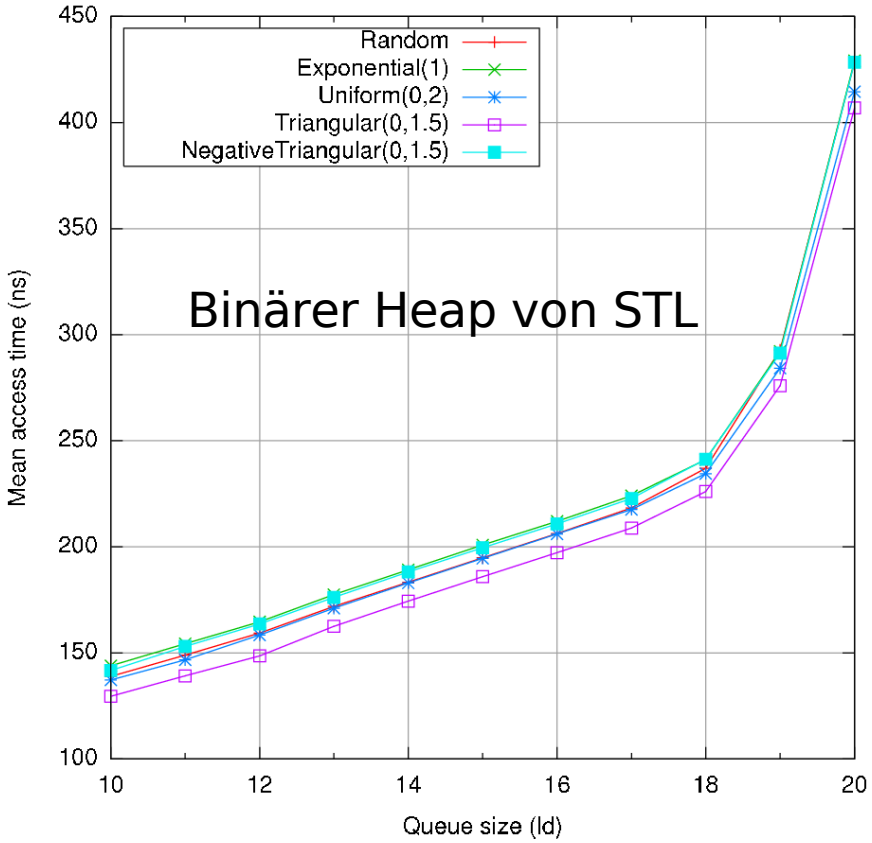


Binärer Heap

- O(log n). Optimal!
- Cache-effizient (impliziter Heap)
- Einfach
- Schnellste allgemeingültige Datenstruktur für Prioritätswarteschlangen

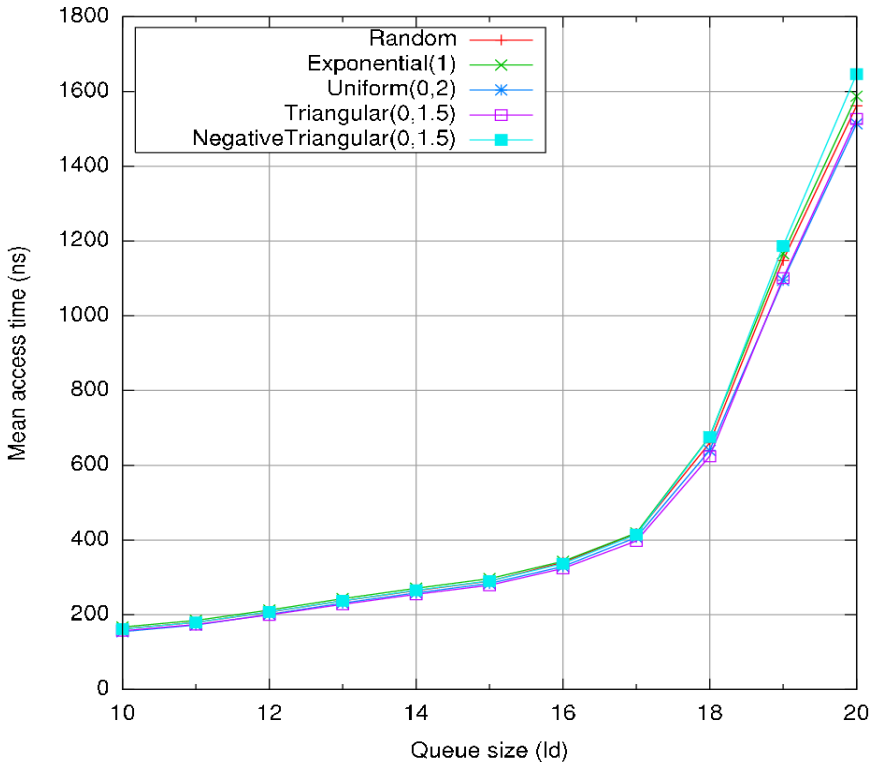
Benchmarks


Random
Exponential(1)
Uniform(0,2)
Triangular(0,1.5)
NegativeTriangular(0,1.5)
Mean access time (ns)
Queue size (ld)
Mein Binärer Heap


Random
Exponential(1)
Uniform(0,2)
Triangular(0,1.5)
NegativeTriangular(0,1.5)
Mean access time (ns)
Queue size (ld)
Binärer Heap von STL

Pairing-Heap

- O(1) insert
- amort. O(log n)
- Zeiger-intensiv!



Calendar Queue

- O(1)
- O(n) w. c.
- Kompliziert!
- Spezialisiert!