FASAN

Eine funktionale automatisch parallelisierbare Sprache für numerische Anwendungen

Ralf Ebner

TU München

1. Warum „funktional“ ?
2. Numerik im funktionalen Stil
3. FASAN–Anwendung: ARESO
4. Projekt: Lastverteilung

Motivation

Problem: Numerische Aufgabenstellungen werden immer komplexer und aufwendiger

$\Rightarrow$ **Parallelisierung** notwendig

Crux der imperativen Programmiersprachen:

- Komplexe Abhängigkeiten zwischen den Programmteilen (globale Variablen, Zeiger)
  $\Rightarrow$ **automatische** Parallelisierung schwierig
- Zwang zur Festlegung der Befehlsreihenfolge
  $\Rightarrow$ **explizite** Parallelisierung unübersichtlich

Abhilfe: Funktionale Programmierung

Was heißt „funktional“ ?

Mathematischer Funktionsbegriff: $y = sin(x)$

- Funktionen beschreiben Abbildung von Eingabe- auf Ausgabe-Daten
  $\Rightarrow$ keine **Seiteneffekte**
- Funktionen sind rechenbereit, wenn Eingabedaten verfügbar sind
  $\Rightarrow$ keine **explizite Ablaufsteuerung**
- Bezeichner erhalten einmal festgelegte Werte; keine Veränderung von „Prozedurvariablen“, keine Aliase
  $\Rightarrow$ kein **globaler Zustand**

Deshalb:

- Übersichtlichere Programme
- Leichtere Korrektheitsbeweise
- Einfachere Parallelisierung

Numerik im funktionalen Stil

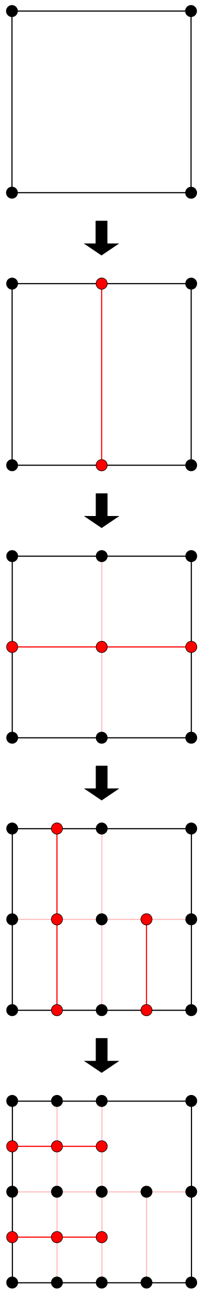
Anforderungen:

- **Effizienz**: hohe Geschwindigkeit, niedriger Speicherbedarf
- **Wiederverwendung** großer Programmteile und Bibliotheken
- **Einfacher** Einsatz

Konsequenzen für FASAN:

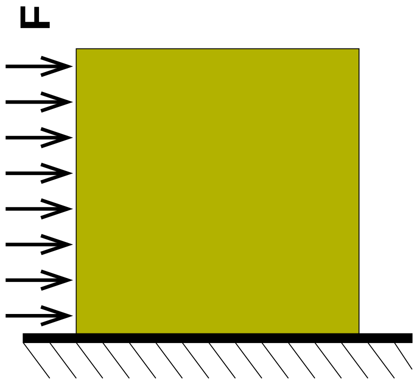
- Einbindung von Funktionen imperativer Sprachen (C, Fortran, ...).
- Als Daten sind Referenzen zugelassen (Arbeiten am Platz).
- Funktionale Ebene dient nur der parallelen Konfiguration der sequentiellen Module.
- Effiziente Implementierung baum-artiger Datenstrukturen

FASAN am Beispiel von ARESO



Divide & Conquer-Algorithmus:
Rekursive Aufspaltung des Problems in Teilprobleme
$\Rightarrow$ Baum von Finiten Elementen

Beispiel:
Kragscheibe



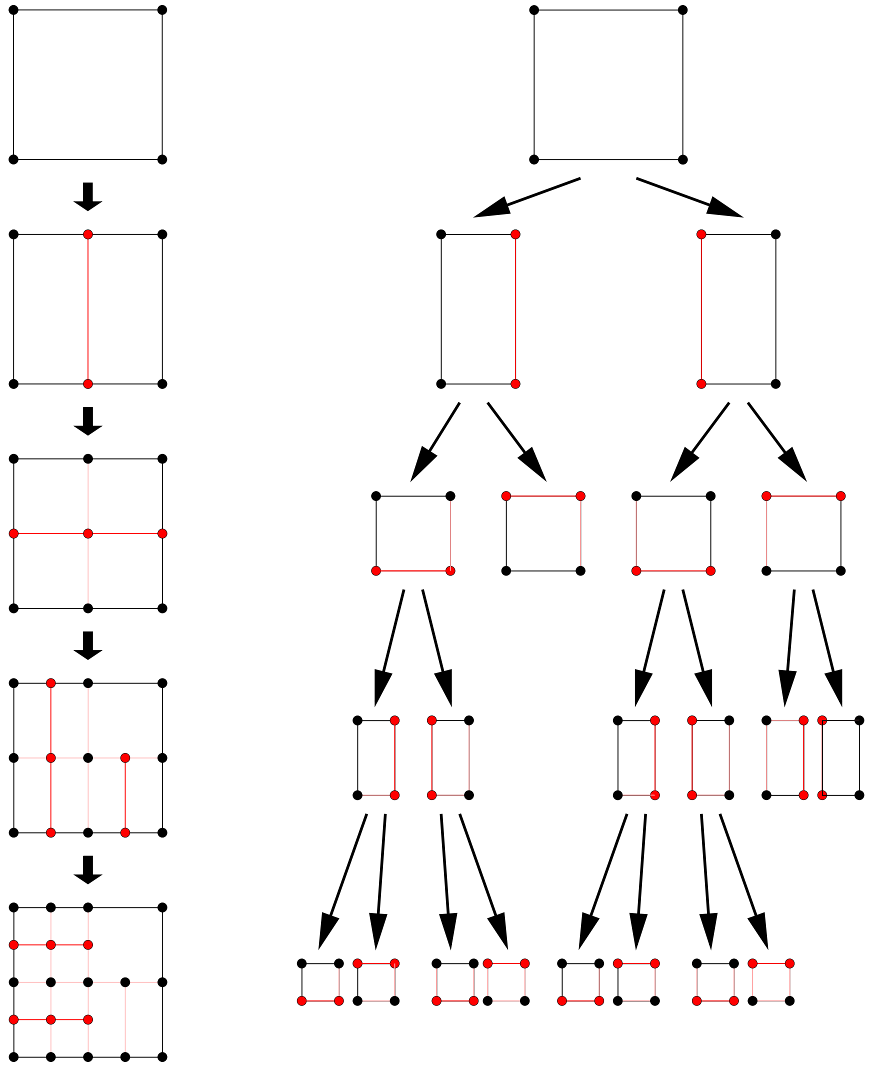
Baum-Durchlauf (W-Zyklus):

- Kondensation der Steifigkeitsmatrix aus den Teil-Elementen
- Rücksubstitution, Fehlerschätzung, Verfeinerung

Iteration des Baum-Durchlaufs, solange Fehler zu groß

ARESO

Substrukturierung und Datenstruktur



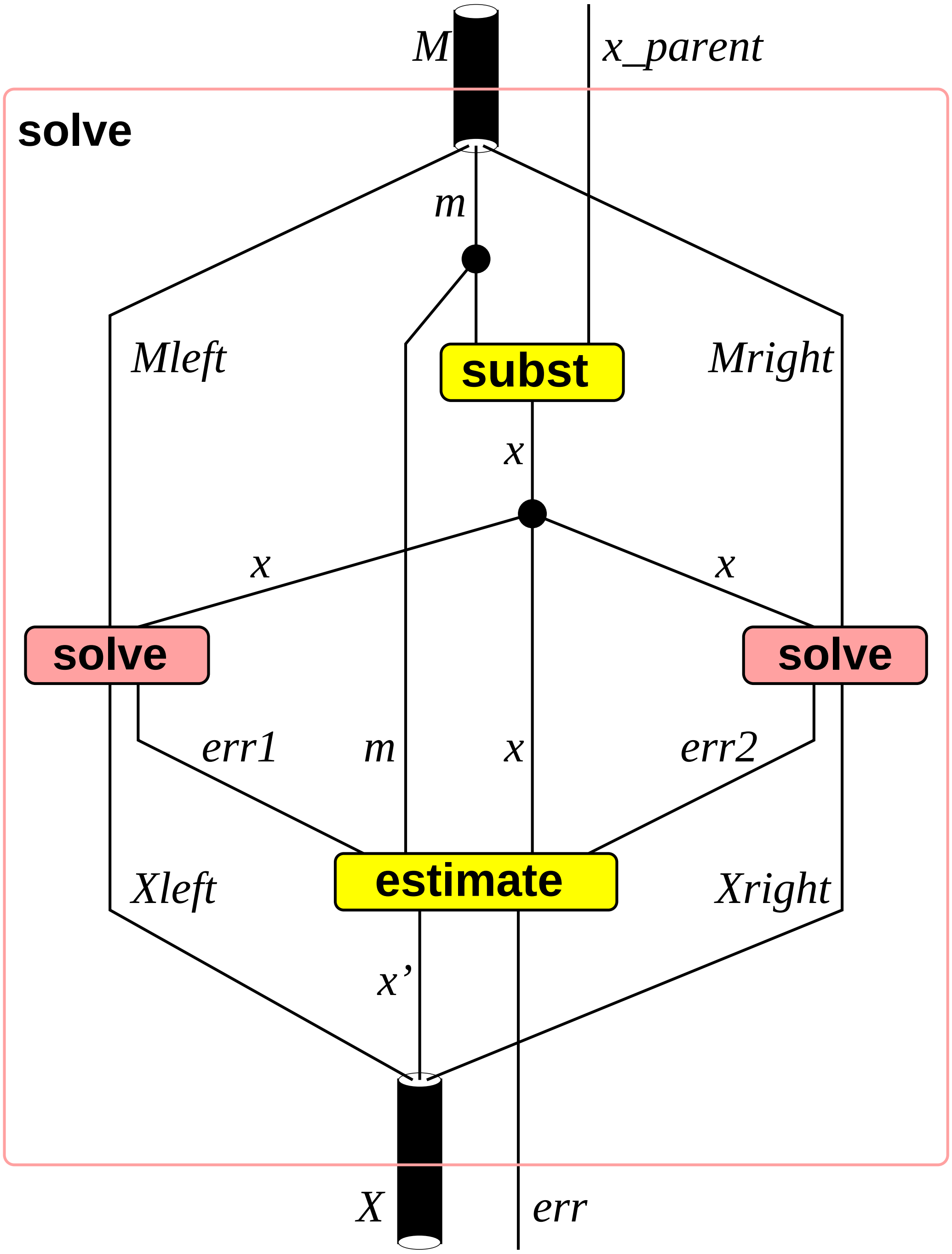
Ziel: Effiziente funktionale Beschreibung

ARESO: FASAN-Programm

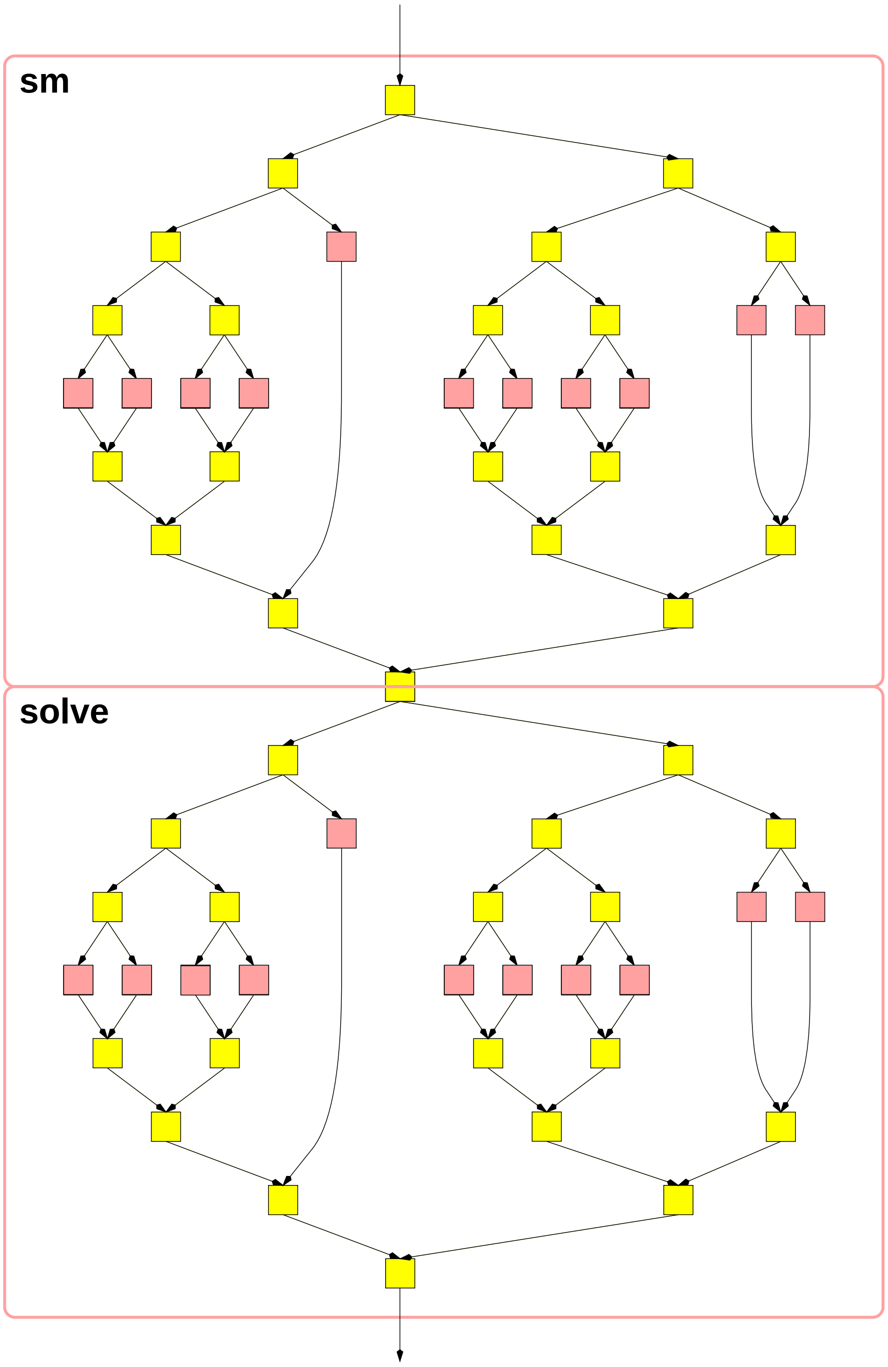
```
agent areso (M, X_start: Tree; eps): X: Tree;
stream
   M', X': Tree;
begin
   M' := sm (M, X_start);
   X', err := solve (M', nodata());
   if error2big (err,eps) then
      X := recycle areso (M', X', eps);
   else
      X := postprocess (X');
   fi
end

agent sm (M: Tree; X: Tree): M': Tree; ...

agent solve (M: Tree; x_parent): X: Tree; err;
begin
   if isNil(M) then
      X, err := nil, null();
   else
      ⟨Mleft,m,Mright⟩ := M;
      x := subst (m, x_parent);
      Xleft, err1 := solve (Mleft, x);
      Xright, err2 := solve (Mright, x) on next_proc;
      x', err := estimate (m, x, err1, err2);
      X := ⟨Xl,x',Xr⟩
   fi
end
```

Datenflußgraph

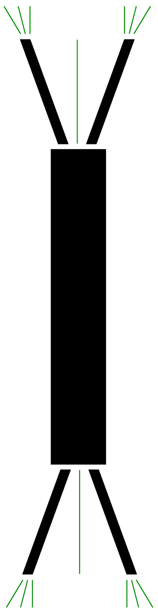
else-Zweig der solve-Funktion

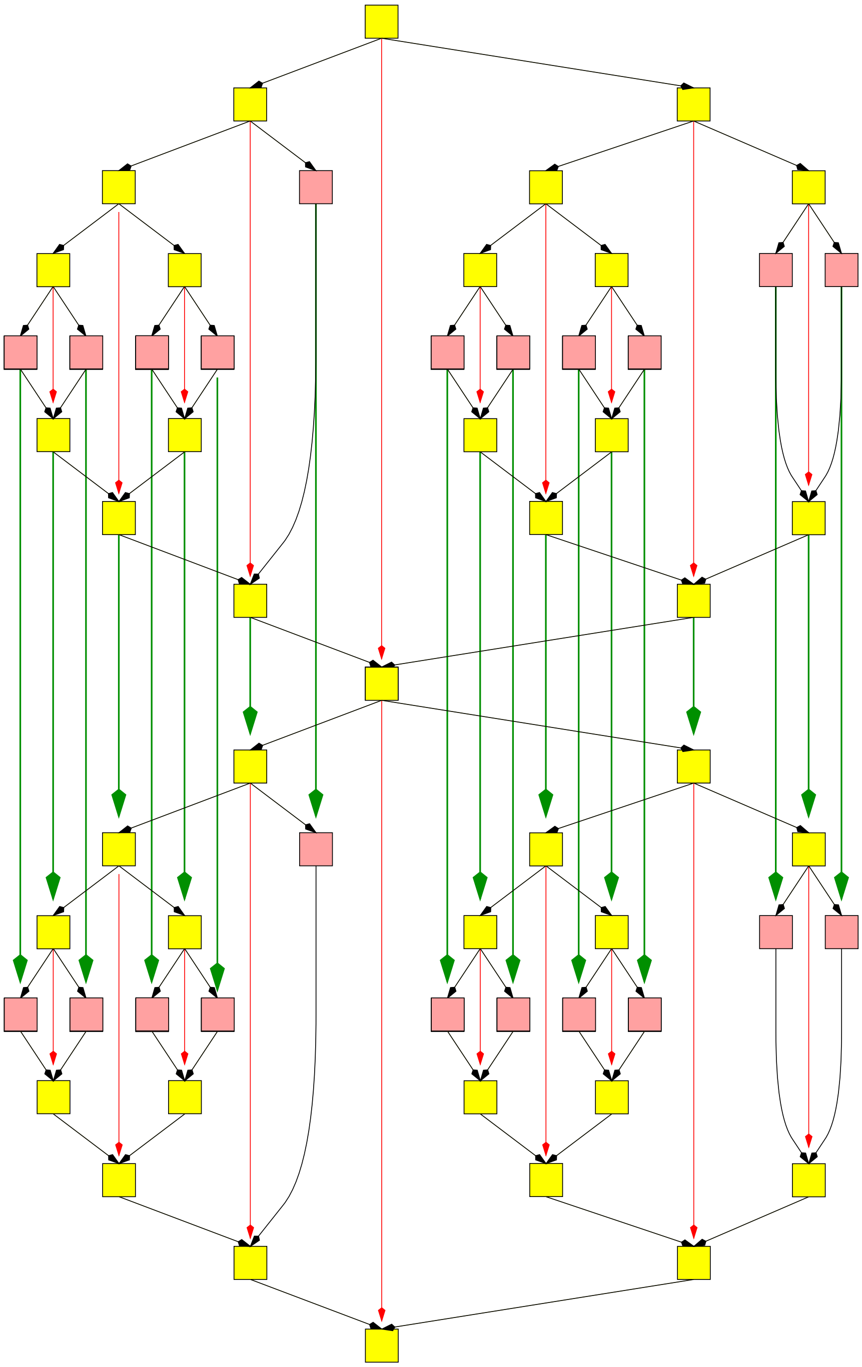


Datenflußgraph eines W-Zyklus



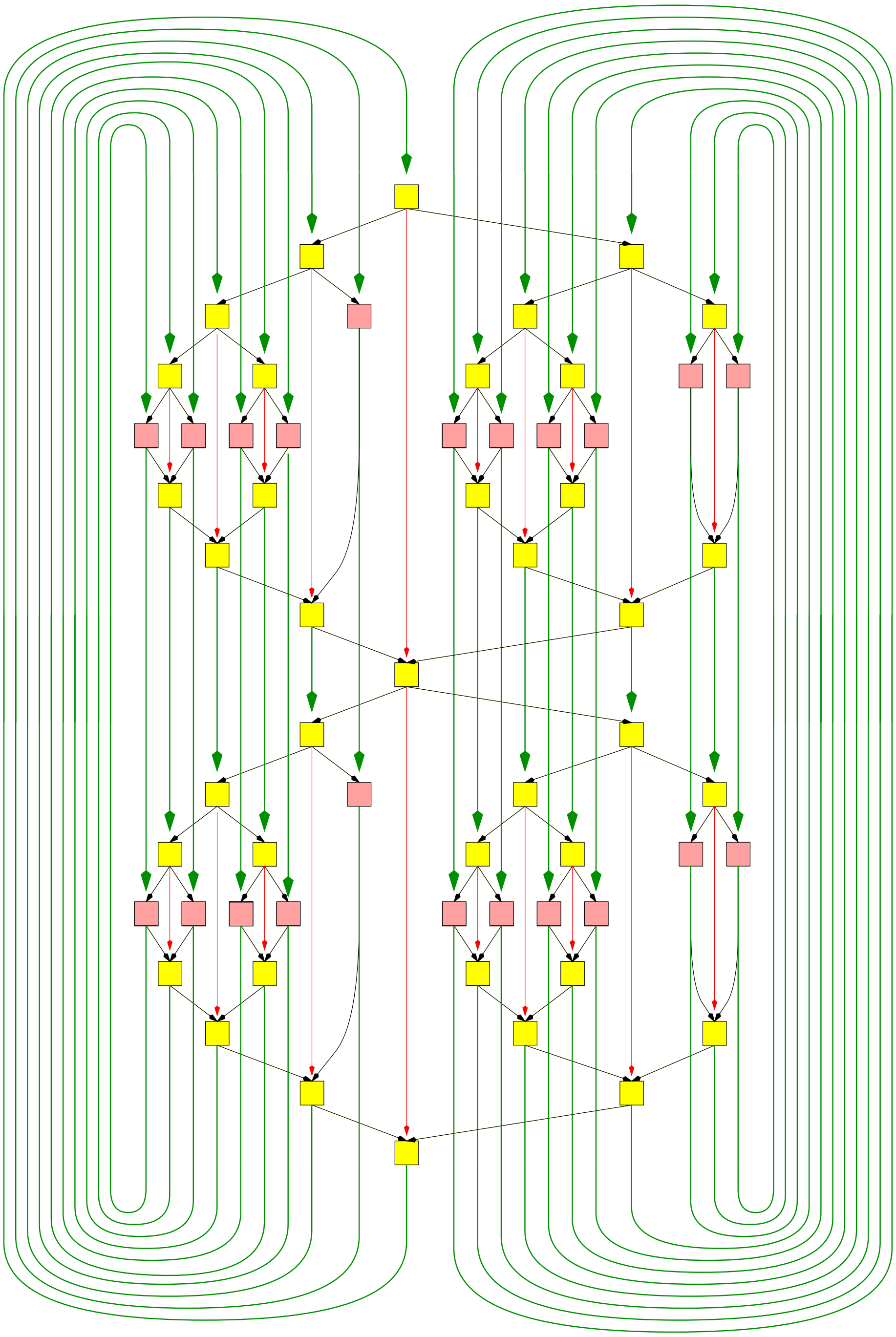
Entflechtete Kabelbäume





Iterationen

Endrekursion $\Rightarrow$ Zyklen im Datenflußgraph



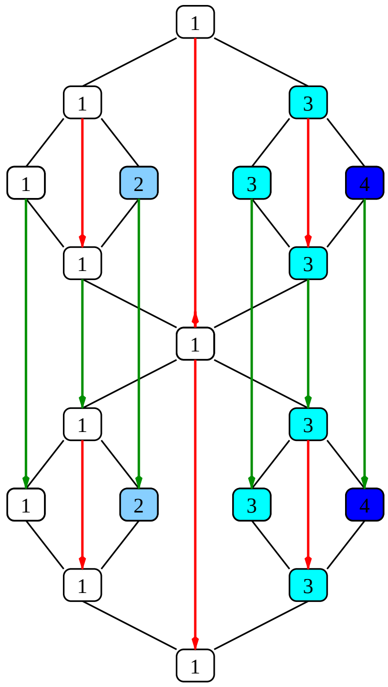
Konzepte zur Lastverteilung

Bisher:

- benutzerdefiniert (Lokationsfunktionen)
- statisch

Ziel:

- automatisch
- dynamisch



Weiterentwicklung:

FASAN als Frontend für **DYNASTY**

(Lastverteilungssystem nach Börsenmodell)

Maklerprozesse verteilen die Berechnungsaufträge dynamisch im Rechnernetz, in Abhängigkeit von

- Benutzerlast
- Auftragsgröße
- Datenvolumen
- Rechnerleistung ...

Zusammenfassung

- Parallelität in funktionalen Programmen kann automatisch ausgenützt werden.
- FASAN konfiguriert sequentielle Routinen aus Standard-Sprachen
- Kabelbäume verhindern unnötige Kommunikation und Synchronisation in baumartigen Datenstrukturen
- **Projekt**: Durch dynamische Lastverteilung laufen die FASAN-Programme völlig automatisch parallel.