Professionelles GPISpace

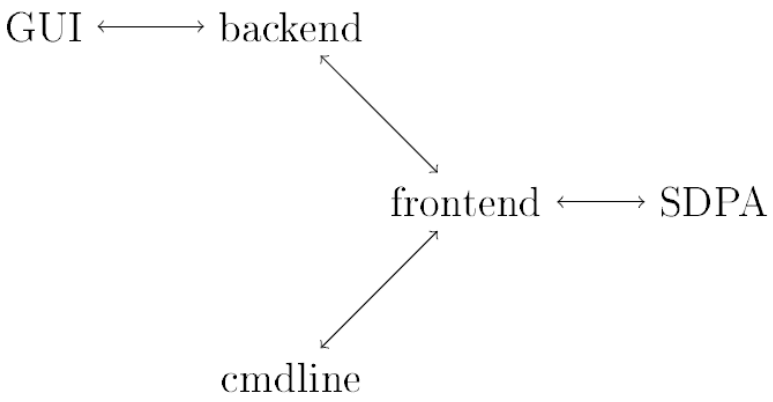
"Meine schlimmsten Schmerzen"

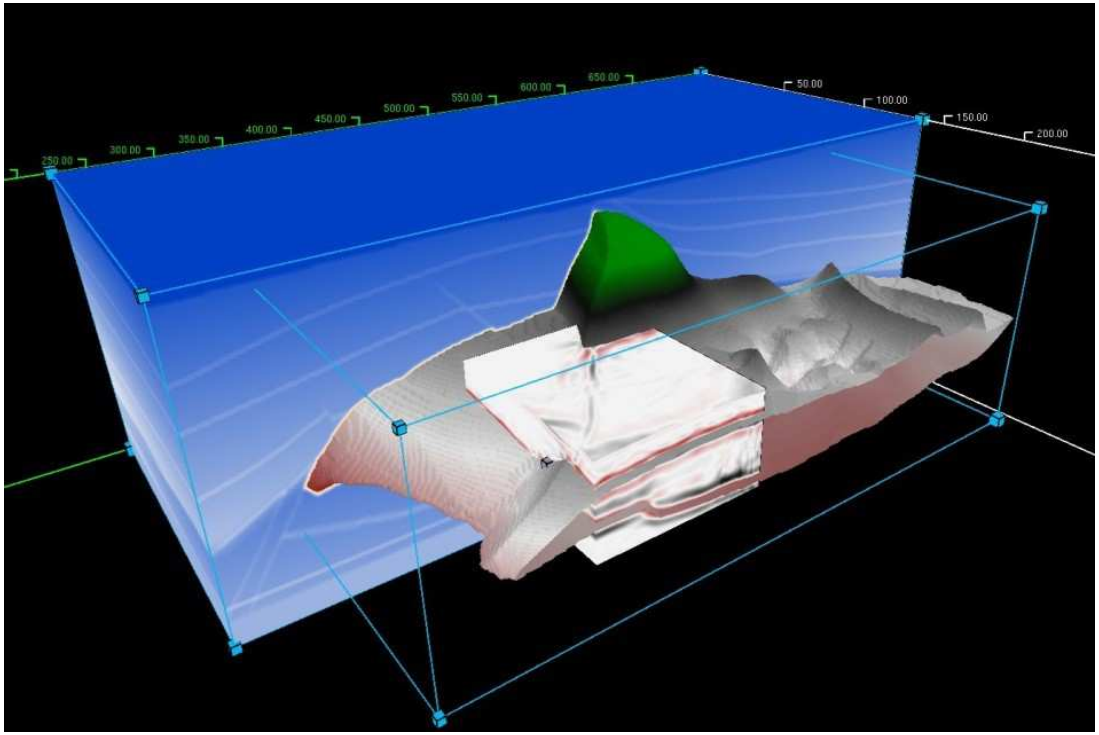
March 13th 2013

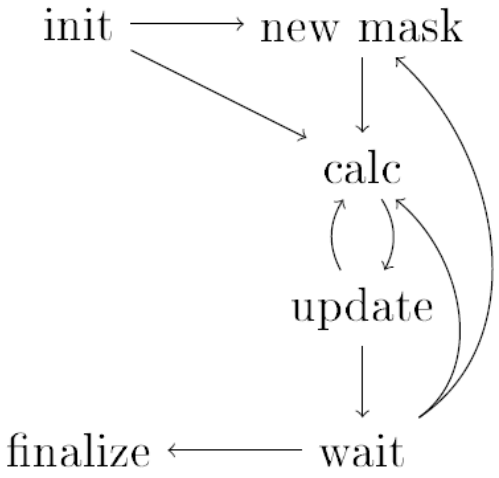
Dirk Merten

Fraunhofer ITWM

ISIM and SDPA: Current Usage







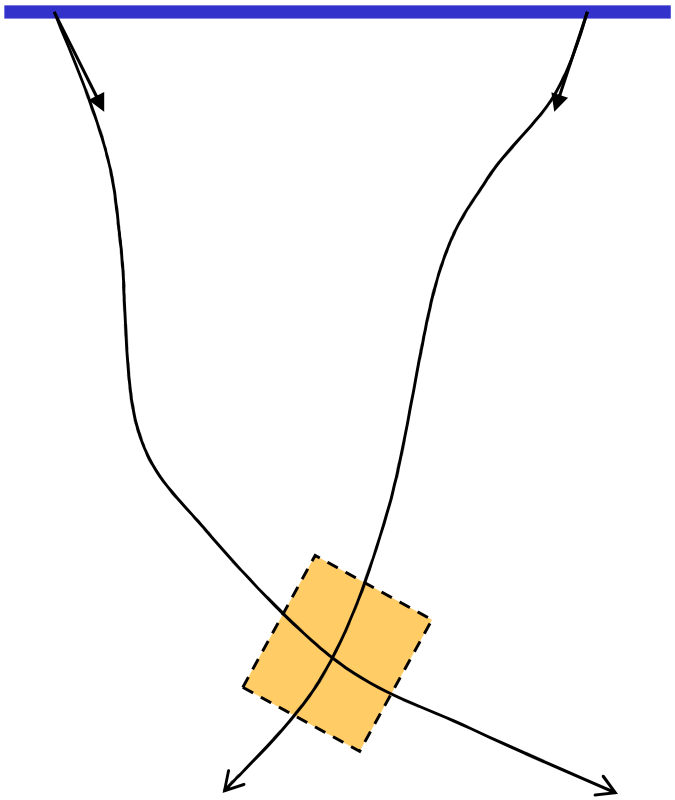
Fraunhofer ITWM

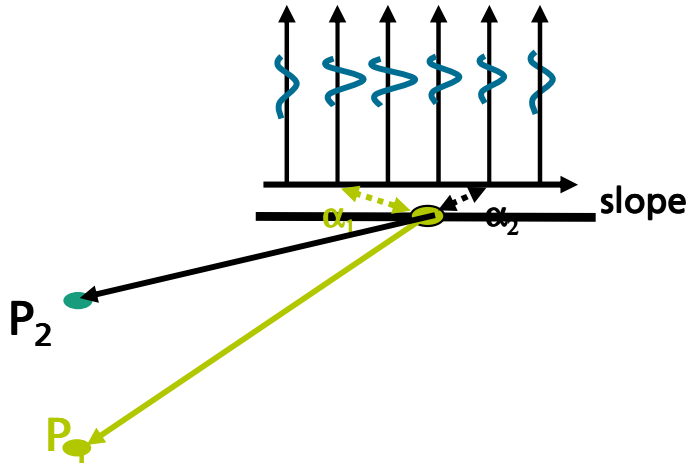
ISIM and SDPA: Current Usage

- Beam Stack Migration and Fast Beam Migration as SDPA applications running at StatOil
- Start-Up by GUI
- or from shell like “normal” binary
- pnets part of start-up script as “here-documents”
- execute pnetput, sdpa init, start, submit and stop
- shared objects locally within application directory
- setting LD_LIBRARY_PATH and PC_LIBRARY_PATH

→ porting SlantStack algorithm to the same run-time system

Parallel Concurrency : Ultra-Fast Beam Migration

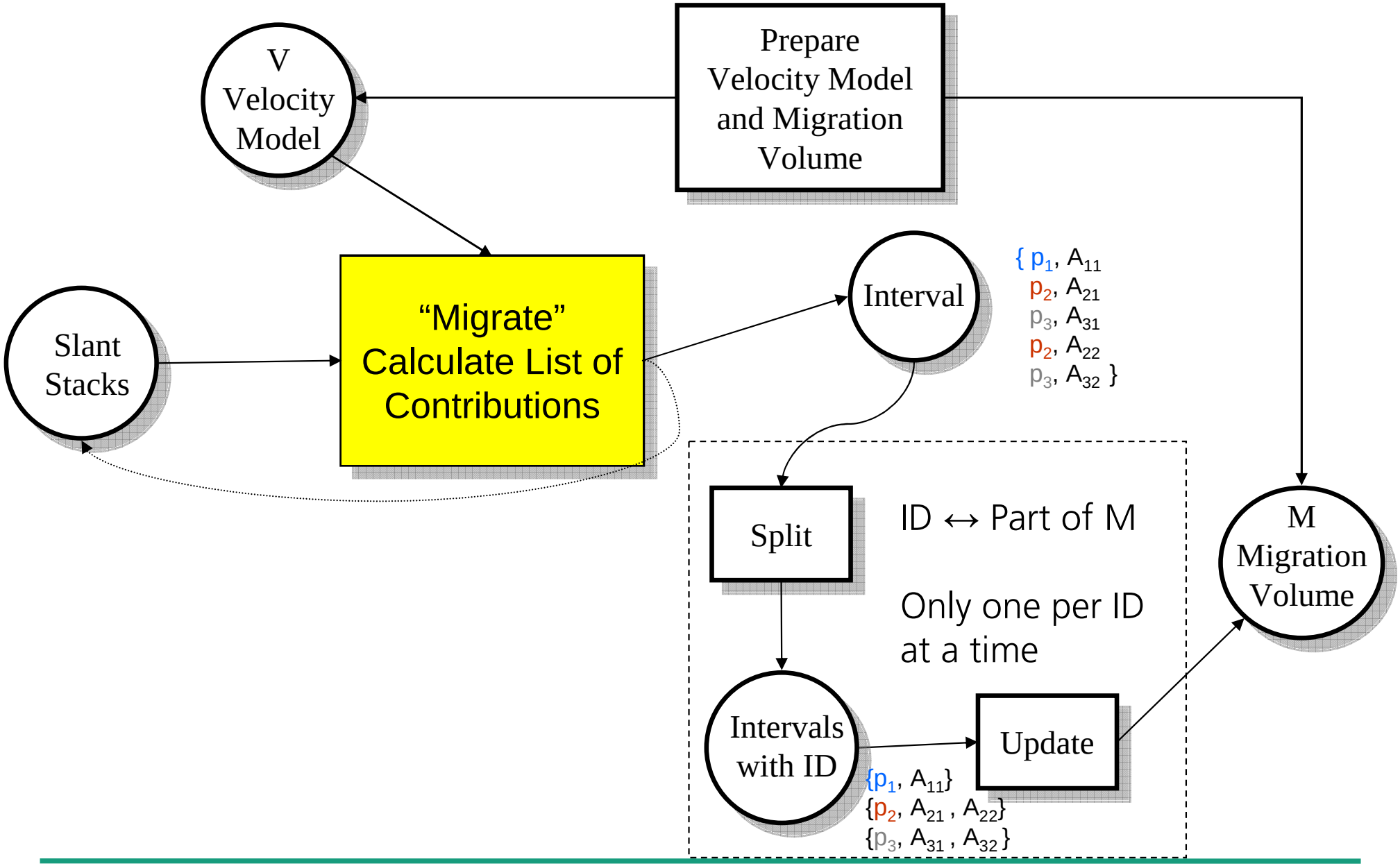




- Ray Tracing leads to small output region affected by slant stack
- output points to be updated are determined during the algorithm
- Parallelization in input and output leads to concurrency during the update

Fraunhofer ITWM

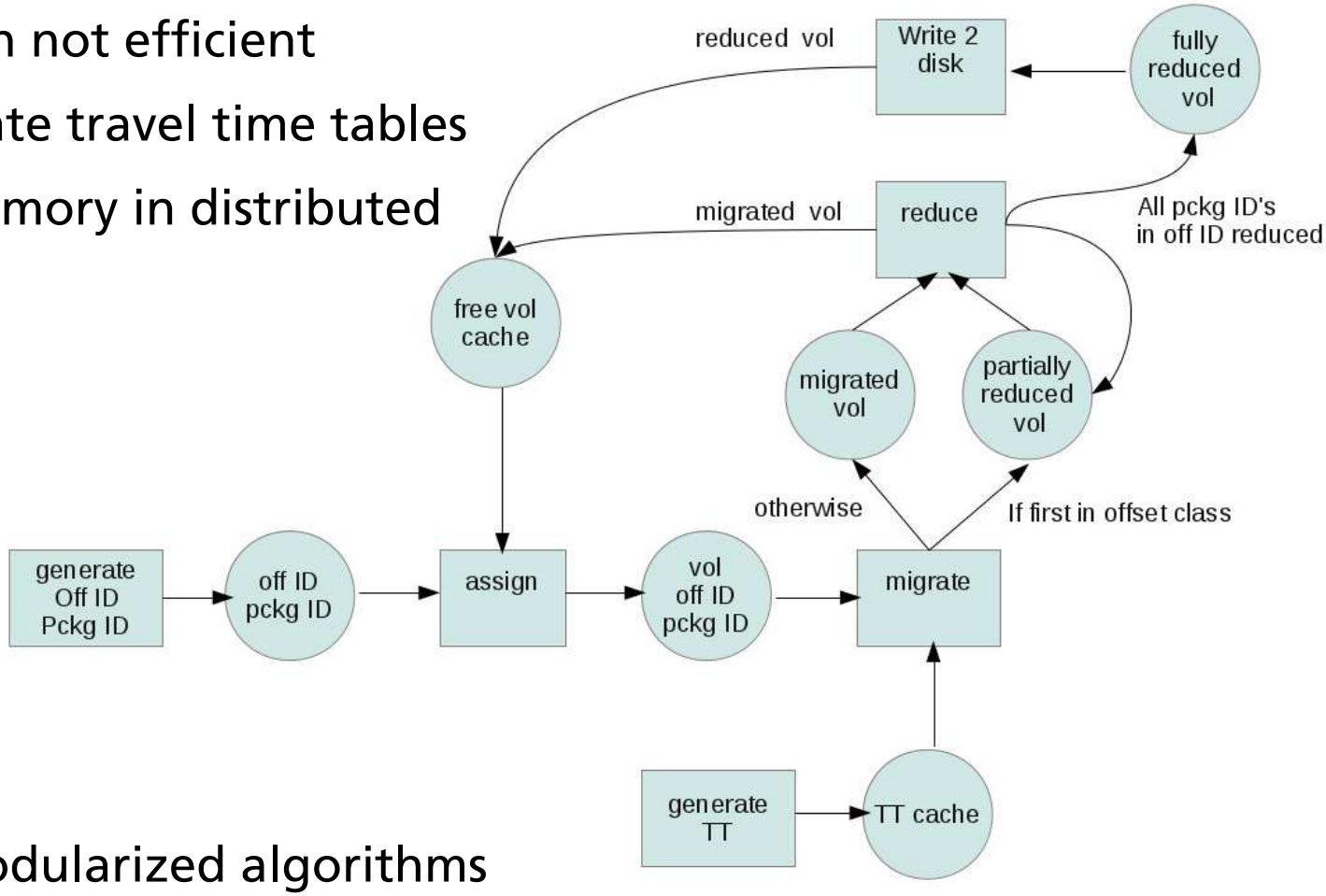
Parallel Concurrency : Fast Beam Migration



Fraunhofer ITWM

Beam Stack Migration

- if event selection not efficient
- Work with / create travel time tables
- Work on full memory in distributed memory caches
- ongoing work



- benefits from modularized algorithms
- Can use “generic” connection to ISIM GUI

Fraunhofer ITWM

Übersicht

- Programmieren
- Kompilieren
- Linken
- Testen
- Debuggen
- Optimieren
- Portieren

- Funktioniert gut
- Umständlich, aber geht
- Unmöglich / Unentscheidbar

Programmieren

- Hilfreicher GPI Memory Manager
- (Schnittstellen-) Code schnell in xml eingefuegt
- Netz Schittstellen sind redundant → viel copy-n-paste
- 3 weitere Ebenen:
  - xml
  - generierter Code, Schnittstelle zwischen Netz-Strukturen und C
  - Modules als Treiber fuer Applikation

```
cat share/sdpa/xml/lib/grid/size.xpnet
<defun>
  <in name="grid" type="grid_type"/>
  <out name="size" type="long"/>
  <expression>
    ${delta_x} := ${grid.ur.x} - ${grid.ll.x};
    ${delta_y} := ${grid.ur.y} - ${grid.ll.y};
    ${sizex} := 1 + long (floor (${delta_x} / ${grid.s.x}));
    ${sizey} := 1 + long (floor (${delta_y} / ${grid.s.y}));
    ${size} := ${sizex} * ${sizey};
  </expression>
</defun>
```

- Code und Expressions aus libs steht nur xml zur Verfügung
  → Code dupliziert

Fraunhofer ITWM

Kompilieren

- aufschlussreiche Fehlermeldungen des pnetc

- Syntax-Fehler in generiertem Code → Suche und Editiere xml
- Fehler in structure : geschrieben, generiert oder aus library ?
- i.A. mehr Browsing

- Syntax-Fehler in Expressions werden beim Komplilieren nicht entdeckt, erst zur Laufzeit (und dann nur kryptisch)
- Make dependencies von xml, generiertem code, externem code alle berücksichtigt ?
- Löscht “make clean” auch wirklich alles, auch generierten code?

Linken

- wird nicht gemacht …
- shared objects:
  Fehlende Dependencies fallen erst zur Laufzeit auf
- Abhängigkeit von externen Libraries:
  Welche Boost-Version brauche ich
  ab welcher SDPA Version bei Intel V x.y.z

```
for i in *.so; do
  nm $i |grep " U “
        |grep -v GLIBC_
        |grep -v GLIBCXX_
        |grep -v CXXABI_
        |grep -v GCC_
        |grep -v "pthread_“
        |grep -v " sem_“
        |grep -v boost
        |grep -v fvm
        |grep -v fhg;
done
```

- Dependencies können mit falschen Symbolen erfüllt werden
  LD_LIBRARY_PATH überall richtig gesetzt?

Testen

- make run:
  Ausführung auf einem Knoten (falls das möglich ist)
- Unit Tests von (Teil-) Netzen
- Unit Tests mit generiertem Code

Debuggen

- Segmentation Faults im GPI Speicher liefert aufschlussreiche Fehlermeldung
- gpish:
  Überpüfen von Speicherbereichen un
  Daten zur Laufzeit und post morten
- extensives logging
- kein Starten im debugger möglich
  → gdb an fhgkernel häengen
- Dyn. Scheduling :
  wer macht meinen Job?
  → alle ausser einem killen

```
# *************** #
#   Allocations   #
# *************** #

             SIZE              TSTAMP NAME
  134217728  2013-03-12 12:49 drts-node229-1-11871-com
  102541824  2013-03-12 12:50 global_volume_memory
   12817728  2013-03-12 12:50 data.output
  315241920  2013-03-12 12:50 loadTTParam.TotTTMemSize
   16777216  2013-03-12 12:52 gpish-gpi-com
       1024  2013-03-12 12:49 data.output_meta
  119516800  2013-03-12 12:49 data.coarse_vel
  119516800  2013-03-12 12:49 data.coarse_eps
  119516800  2013-03-12 12:49 data.coarse_dlt
 2958312407  2013-03-12 12:49 drts-node229-1-11871-shm
   16777216  2013-03-12 12:52 gpish-shm-com
```

- Debuggen von Expressions ist nicht möglich
- Logging für Anwender zu unübersichtlich

Optimieren

- ´Execution Monitor´:
  Besseres Monitoring als bei normalen Applikationen

- neue bottle necks / penalties:
  z.B. Expression → Module Call

- Probleme im Scheduling kaum zu lokalisieren
- ´Expressions´ nicht erreichbar

Portieren

- Bei strukturiertem Code:
  - Tausche Kommunicationslayer aus
  - Rufe Klassen/Funktionen als Modules auf
- Bei unstrukturiertem Code:
  - Aufwendiges Refakturing und Modularisierung
  - `Zustand` von Instanzen wiederherstellen oder abspeichern
    Bsp.: Progress Bar
    Wer? Wann? Wie weit?
- Einbinden in existierende Build Umgebung ist kompliziert
- Memory Requirement, die sich zur Laufzeit ergeben (nach Start der Applikation), koennen nicht berücksichtigt werden

Wünsche

- Kompilieren / Syntax-Check von Expressions
- Debuggen von Expressions
  - Optional als code generieren
  - Trace logging von allen Operationen
- globale statische structures / atomic counters
- Fehlermeldungen beim Kompilieren verweisen auf xml anstatt auf den generierten Code (so dass emacs dorthin springen kann)
- Monitoring von internem Scheduling
- Starten / Submit aus Code heraus: “C-Schnittstelle”
- Logging nachträglich in GUI aufbereiten
- ? generierten Code über GUI eingeben aber „normal“ verwalten ?

- sichereres Dependency-Tracking (als) in make
- abgesichertes dynamisches Linken, Namespaces