technische universität dortmund

SFB 876 Providing Information by Resource-Constrained Data Analysis

lehrstuhl physik e5

Stack and Memory Management

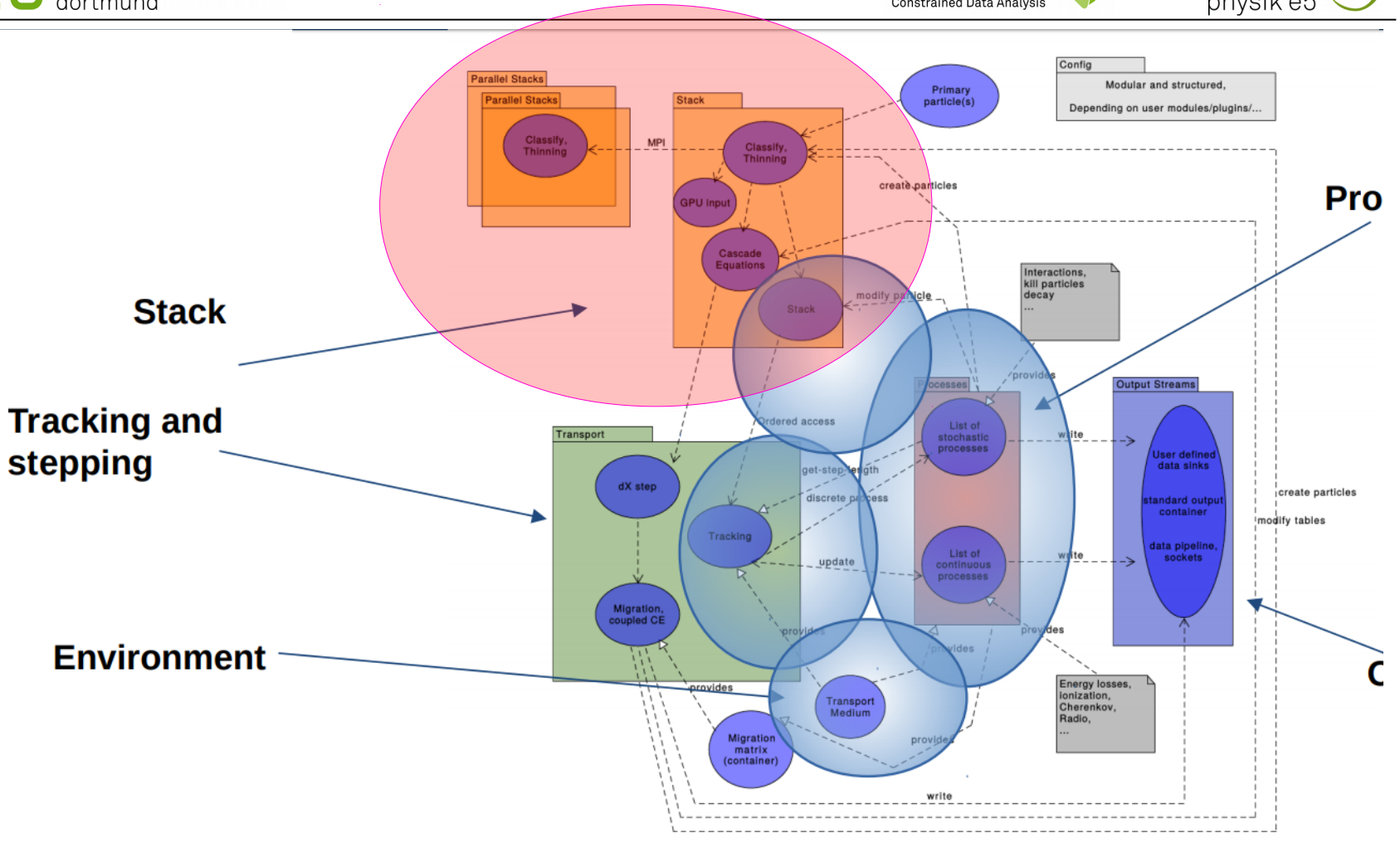
Runtime reduction

Dominik Baack

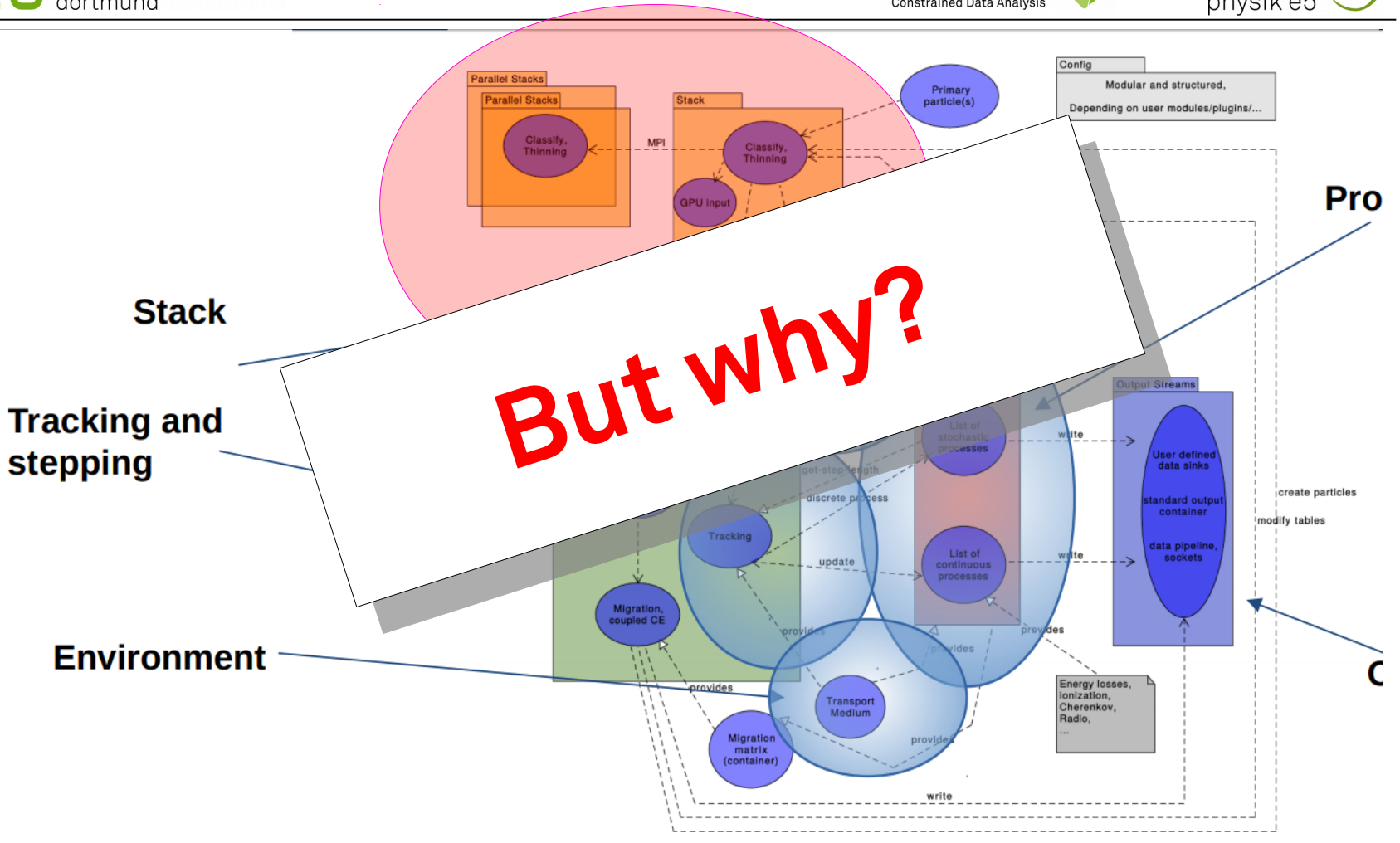
Corsika Workshop

Karlsruhe

20.06.2019

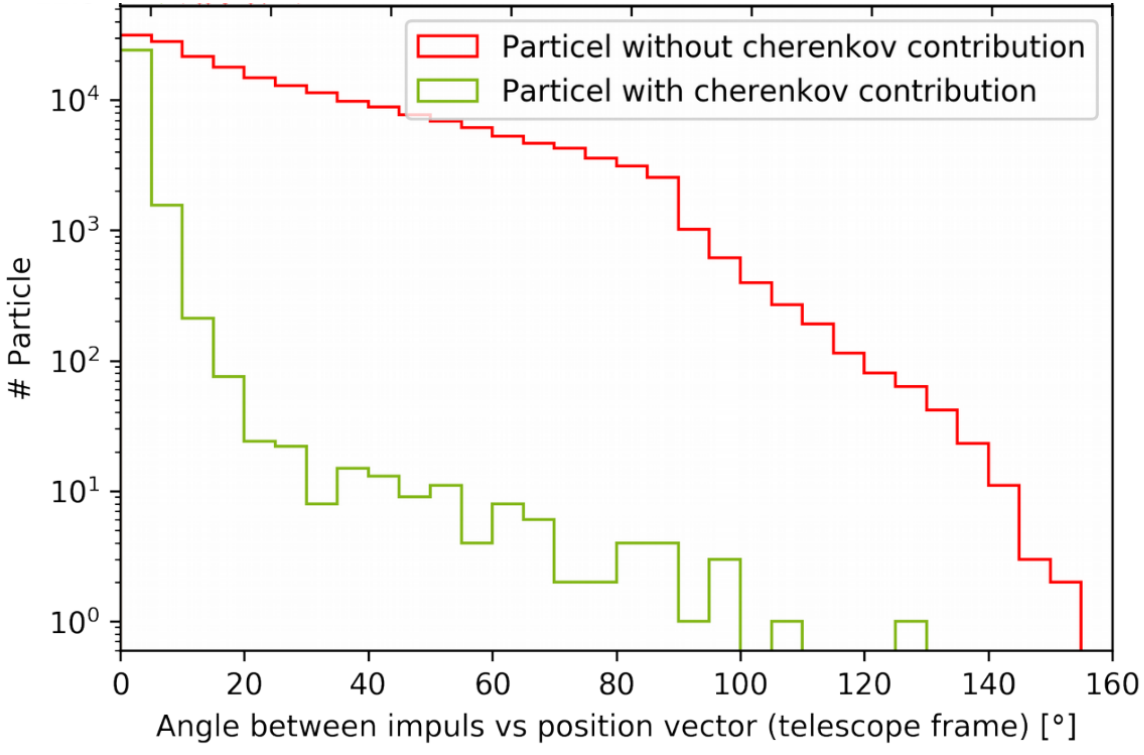


Dominik Baack
dominik.baack@tu-dortmund.de


Stack
Tracking and stepping
Environment
But why?
Pro
C
Parallel Stacks
Parallel Stacks
Classify, Thinning
MPI
Stack
Classify, Thinning
GPU input
Primary particle(s)
Config
Modular and structured,
Depending on user modules/plugins/...
Output Streams
User defined data sinks
standard output container
data pipeline, sockets
List of stochastic processes
write
get-step-length
discrete process
Tracking
update
List of continuous processes
write
create particles
modify tables
Migration, coupled CE
provides
provides
provides
provides
Transport Medium
Migration matrix (container)
Energy losses, Ionization, Cherenkov, Radio, ...
write

Target Goal

- Speed/performance increase in Corsika for a broad range of experiments



Target Goal

- Interface for threading without changes to core routines

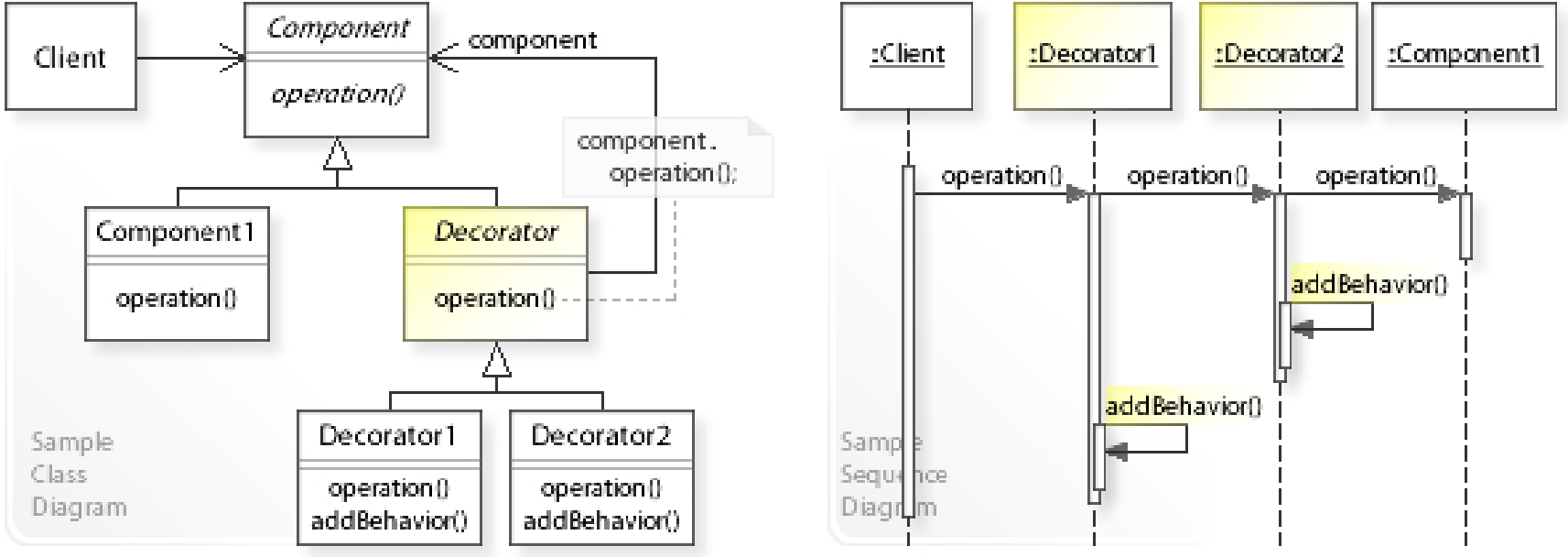
```
/**
 * The Run function is the main simulation loop, which processes
 * particles from the Stack until the Stack is empty.
 */
void Run() {
  SetNodes();

  while (!fStack.IsEmpty()) {
    while (!fStack.IsEmpty()) {
      auto pNext = fStack.GetNextParticle();
      Step(pNext);
      fProcessSequence.DoStack(fStack);
    }
    // do cascade equations, which can put new particles on Stack,
    // thus, the double loop
    // DoCascadeEquations();
  }
}
```

Dominik Baack
dominik.baack@tu-dortmund.de

Dynstack

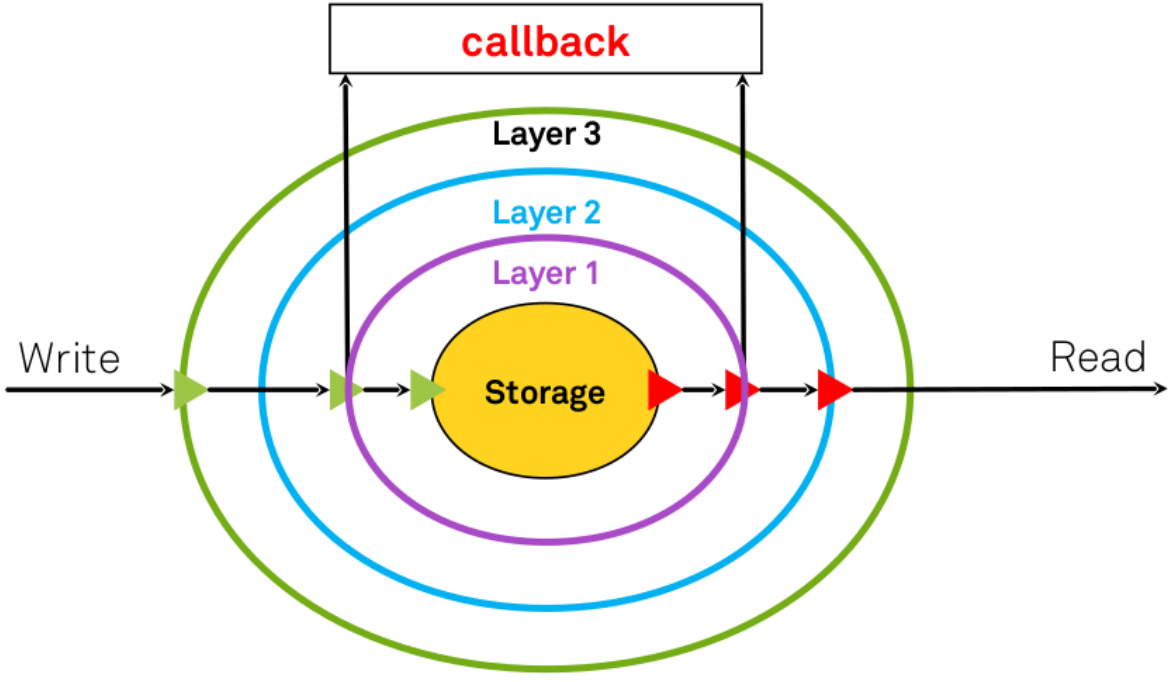
- Decorator Design Pattern similar to Cor++ Dynstack



https://en.wikipedia.org/wiki/Decorator_pattern#/media/File:W3sDesign_Decorator_Design_Pattern_UML.jpg

Dynstack

- Decorator Design Pattern similar to Cor++ Dynstack



```
template <class TType, int TSize=500>
        class storage_fixed : private STACK {
         std::array<TType, TSize> m_data{};
protected:
         using TStorageType = Ttype;
public:
         constexpr storage_fixed() {}

        template <template <class> class T, class ... TArgs>
        constexpr T<storage<TType>> operator<<(T<TType, TArgs> rhs) ...

        constexpr storage_fixed(const storage_fixed<TType>& rhs) ...

        // Copy internaly
        template <template <class> class T, class... TArgs>
        constexpr T<storage_fixed<TType> > addWrapper(TArgs... ctor) const {
                return T<storage_fixed<TType> >(*this, ctor...); }
```

```
template <class Tnext>
class wrapper2 : protected TNext, private STACK {
protected:
        using TStorageType = typename Tnext::TstorageType;
        const int m_offset;
public:
        constexpr wrapper2(const TNext& next, int offset)
                : TNext(next), m_offset(offset) {}

        template <template <class> class T, class... Targs>
        constexpr T<wrapper2<TNext> > addWrapper(TArgs... ctor) {
                return T<wrapper2<TNext> >(*this, ctor...);}

        TStorageType get(int idx) const { return TNext::get(idx) + m_offset; }

        void set(TStorageType data, int idx) { TNext::set(data, idx); }

};
```

Dynstack – Corsika7.6

```
::dynstack::storage::FIFO_STACK<float>
stack(stackSize);

::dynstack::wrapper::InputFilterStack<
      ::dynstack::wrapper::DiscreteSortedStack<
            ::dynstack::wrapper::LIFO_Stack<float>,
      3, sort>,
filter> stack(binSize1, binSize2, binSize3);
```

- Highly customizable
- Removal of unnecessary particles with custom filter
- Prioritization of important particles

Dynstack – CORSIKA 8

- C++17 (type deduction, constexpr) allows easier definition

```
auto storage4_1 =
storage_fixed<int, 5000>().addWrapper<offset>(42);
```

```
auto storage4_1 =
storage_fixed<int, 5000>() << offset(42);
```

?

```
auto storage4_1 =
make_stack<int>(storage_fixed<5000>(), offset(42))
```

?

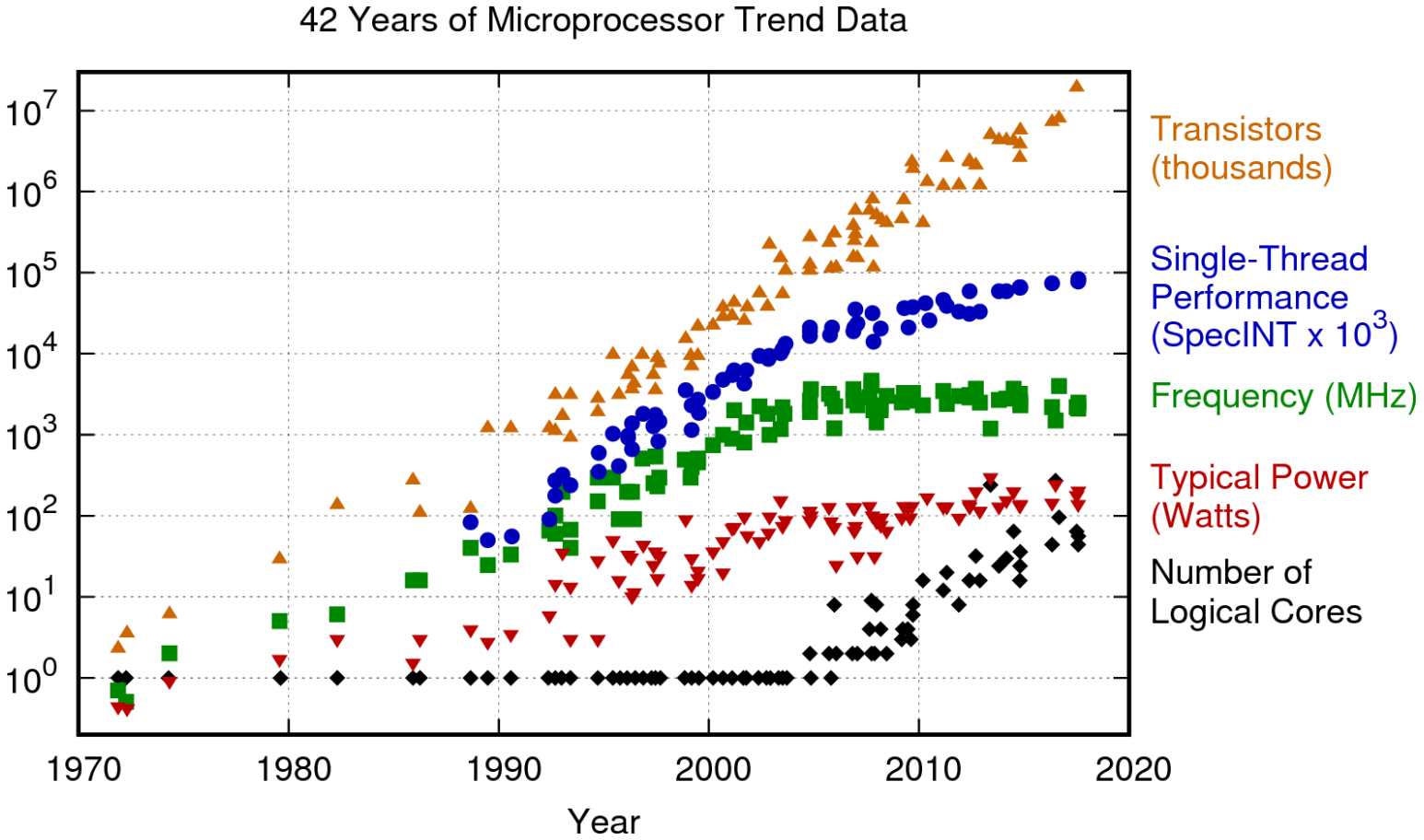
Dynstack – CORSIKA 8

- Dependencies between particle and stack itself
  → „New stack” for every storage type
  → No clear separation between storage and function

- Complexity of saved things is moved into a particle class
  not necessary to implement/change get&set functions inside the stack
  → Complexity of storage and included data is separated

Parallel Computing

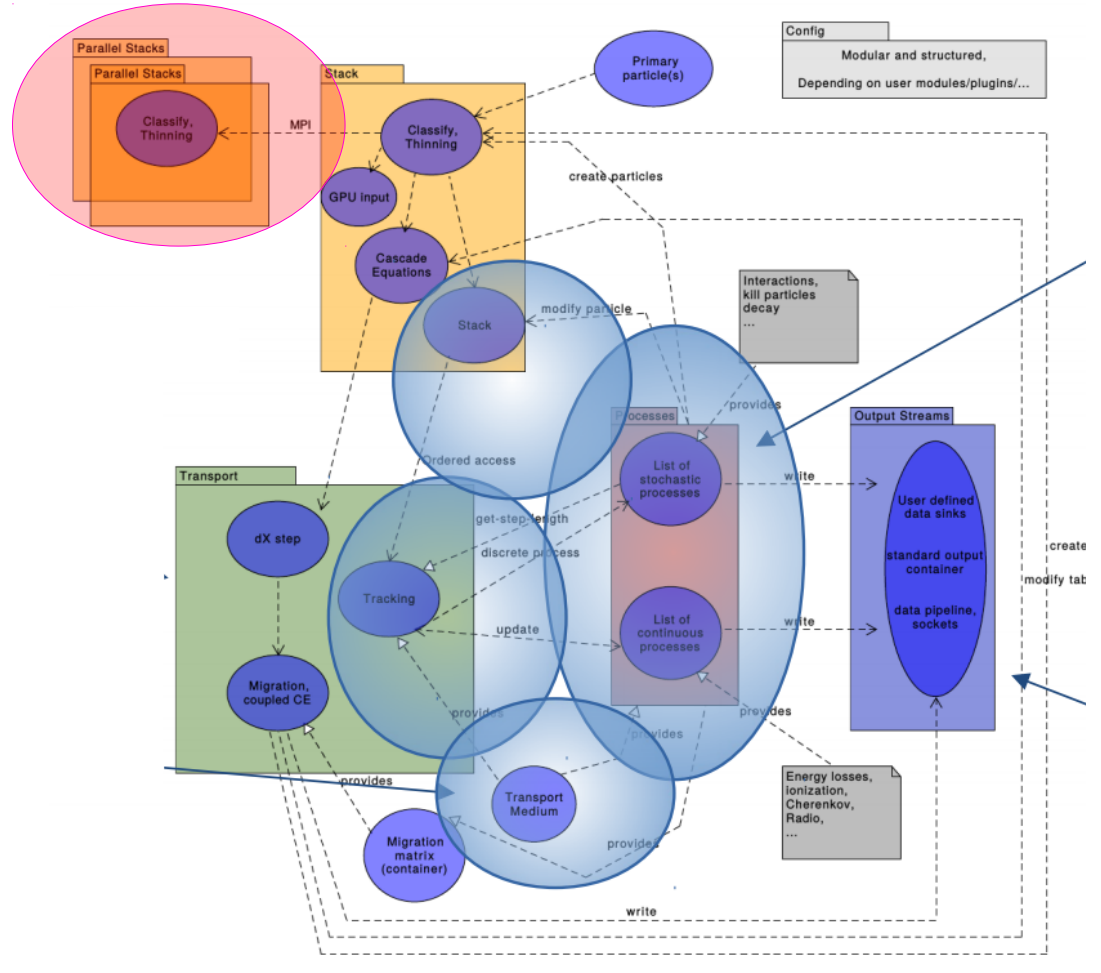
Parallel Computing



Original data up to the year 2010 collected and plotted by M. Horowitz, F. Labonte, O. Shacham, K. Olukotun, L. Hammond, and C. Batten
New plot and data collected for 2010-2017 by K. Rupp

Stack features – Parallel Computing

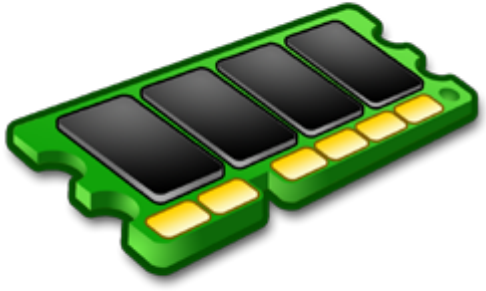
- Can manage write synchronization for multiple threads
- Possible to synchronize over multiple nodes as well



Dominik Baack
dominik.baack@tu-dortmund.de

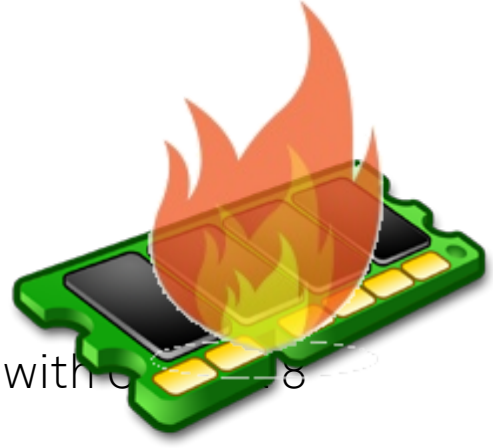
Parallel Computing

- “Physic parallelization”
  → Execute Software X-Times



Parallel Computing

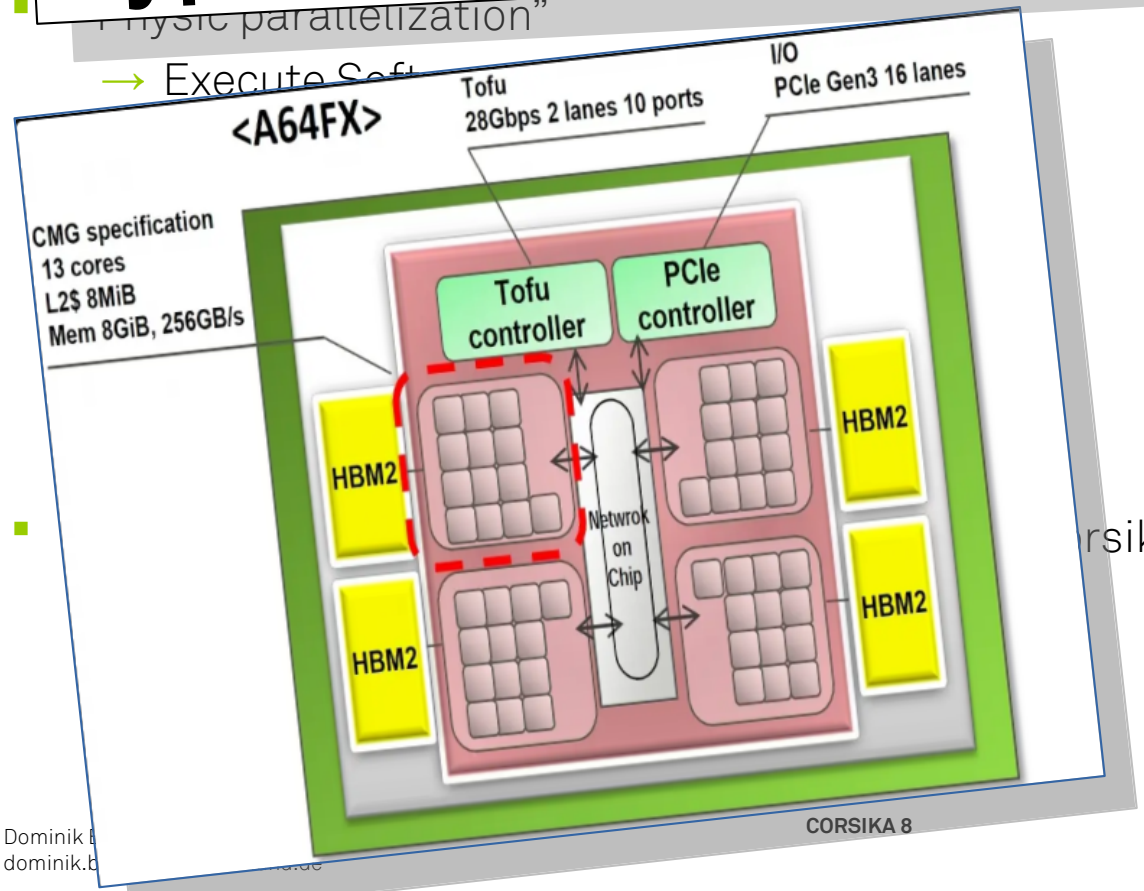
- “Physic parallelization”
  → Execute Software X-Times

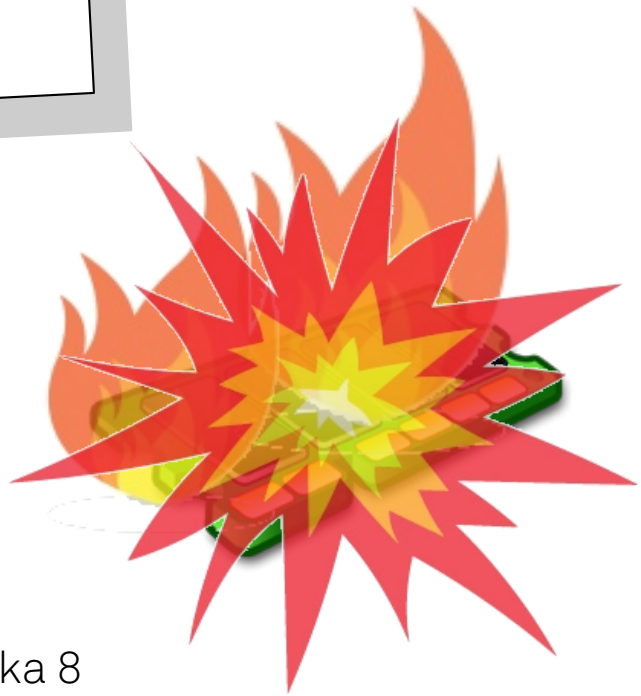


- But: Simulation complexity is massively increasing with C[illegible]8
  → Multiple interaction-models & Environments
  → Complex and Bigger experiments
  → Radio
  → Optimizations
  → ...

Hyper Threading

- Physic parallelization"
  - → Execute Soft
- rsika 8





Parallel Computing

- Common blocks non state free code makes execution with several thread (single process) hard

- Corsika in itself should be mostly state free
  → strong separation between storage and functionality

Parallel Computing - Memory

- Interaction models mostly Fortran
  - → Automatic replacement of common blocks with custom C++ memory management
  - → Introduce paging for non constant memory
  - → Memory mapped files/shared memory between processes