TECHNISCHE UNIVERSITÄT DRESDEN

DRESDEN concept

ZIH
Center for Information Services and High Performance Computing

Score-P

Beyond the Infinite HPC-Domain

Bert Wesarg

Scalable Tools Workshop, 2023

Score-P

Infrastructure for instrumentation and performance measurements

Instrumented application can be used to produce several results:

- Call-path profiling: CUBE4 data format used for data exchange
- Event-based tracing: OTF2 data format used for data exchange

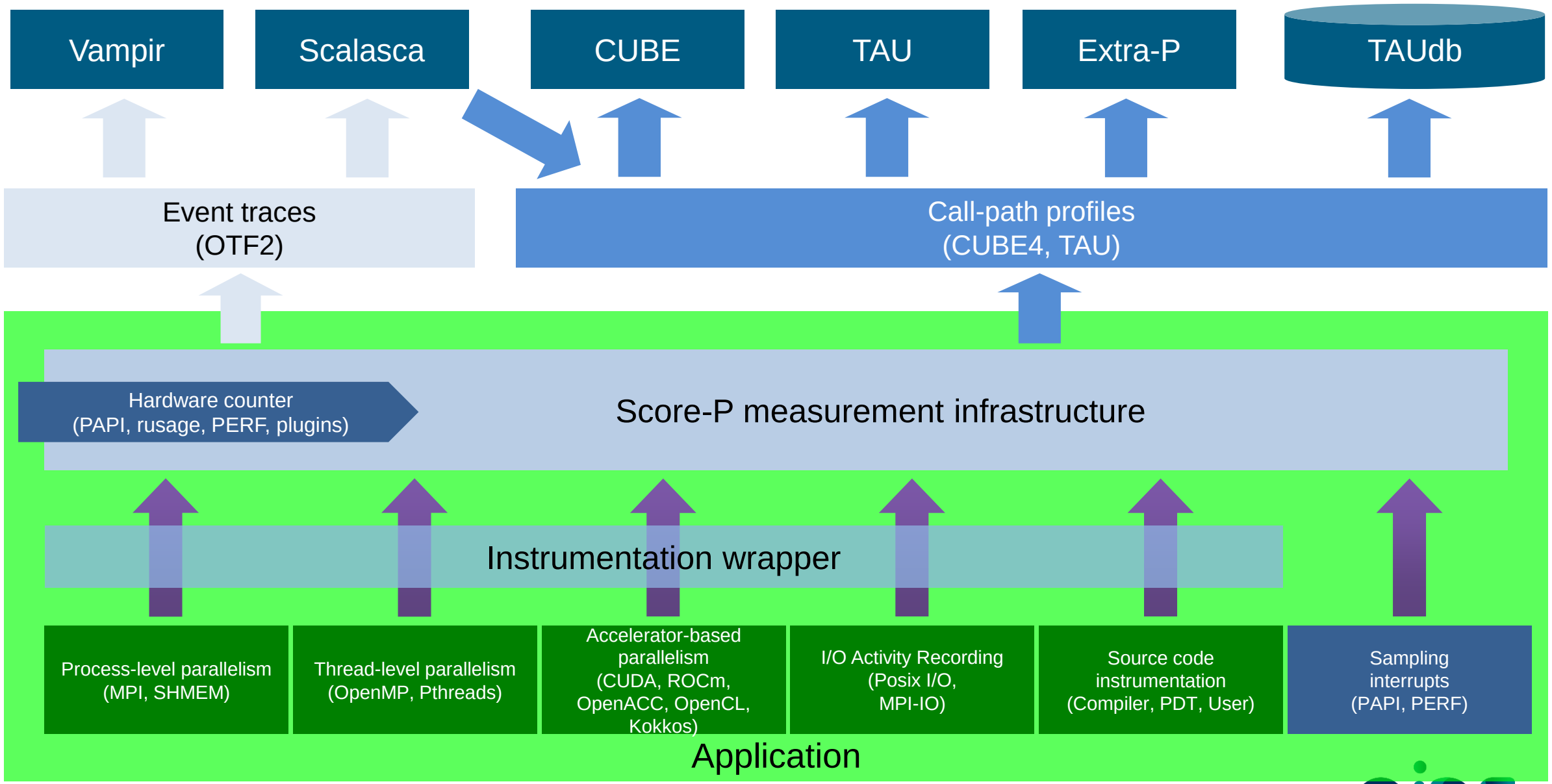
Supported parallel paradigms:

- Multi-process: MPI, SHMEM
- Thread-parallel: OpenMP, Pthreads
- Accelerator-based: CUDA, ROCm, OpenCL, OpenACC

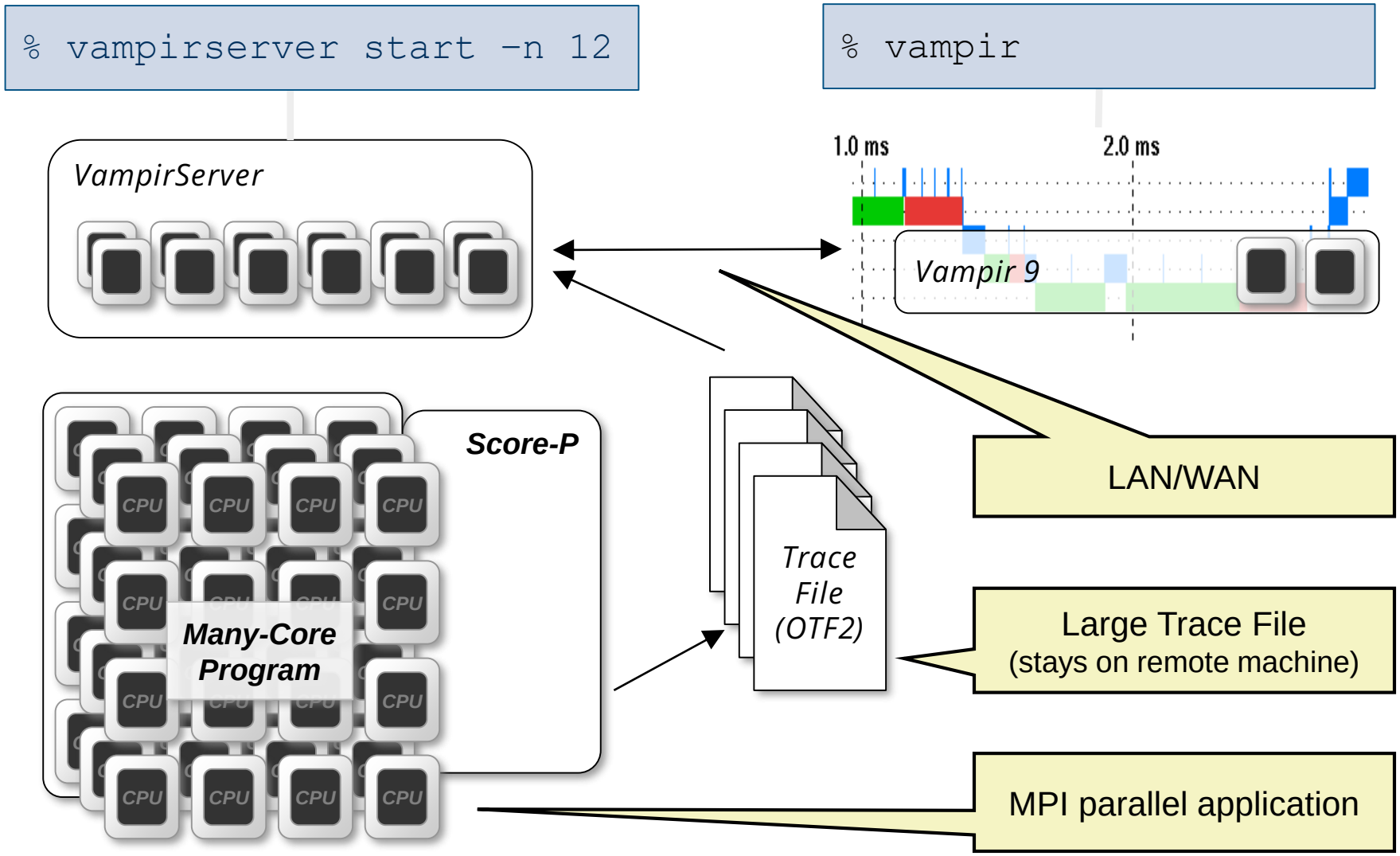
Open Source; portable and scalable to all major HPC systems

Initial project funded by BMBF and close collaboration with PRIMA project funded by DOE

GEFÖRDERT VOM
Bundesministerium für Bildung und Forschung


Vampir
Scalasca
CUBE
TAU
Extra-P
TAUdb
Event traces
(OTF2)
Call-path profiles
(CUBE4, TAU)
Hardware counter
(PAPI, rusage, PERF, plugins)
Score-P measurement infrastructure
Instrumentation wrapper
Process-level parallelism
(MPI, SHMEM)
Thread-level parallelism
(OpenMP, Pthreads)
Accelerator-based
parallelism
(CUDA, ROCm,
OpenACC, OpenCL,
Kokkos)
I/O Activity Recording
(Posix I/O,
MPI-IO)
Source code
instrumentation
(Compiler, PDT, User)
Sampling
interrupts
(PAPI, PERF)
Application

Vampir


% vampirserver start -n 12
% vampir
VampirServer
1.0 ms
2.0 ms
Vampir 9
Score-P
CPU
Many-Core Program
Trace File (OTF2)
LAN/WAN
Large Trace File
(stays on remote machine)
MPI parallel application

The HPC Domain

- "tightly coupled application working on solving the same problem"
- Processes uses a well known communication library (MPI)
- Processes starts collectively
- Processes share a file system

- Score-P requires all properties of an HPC application to record performance data

Outside HPC Domain

- Big Data applications
  - Java based
  - No MPI
  - Non-POSIX file systems
- AI/ML frameworks
  - Python based
  - Custom socket communication
- Workflow frameworks
  - Non-collective start of applications

Previous work

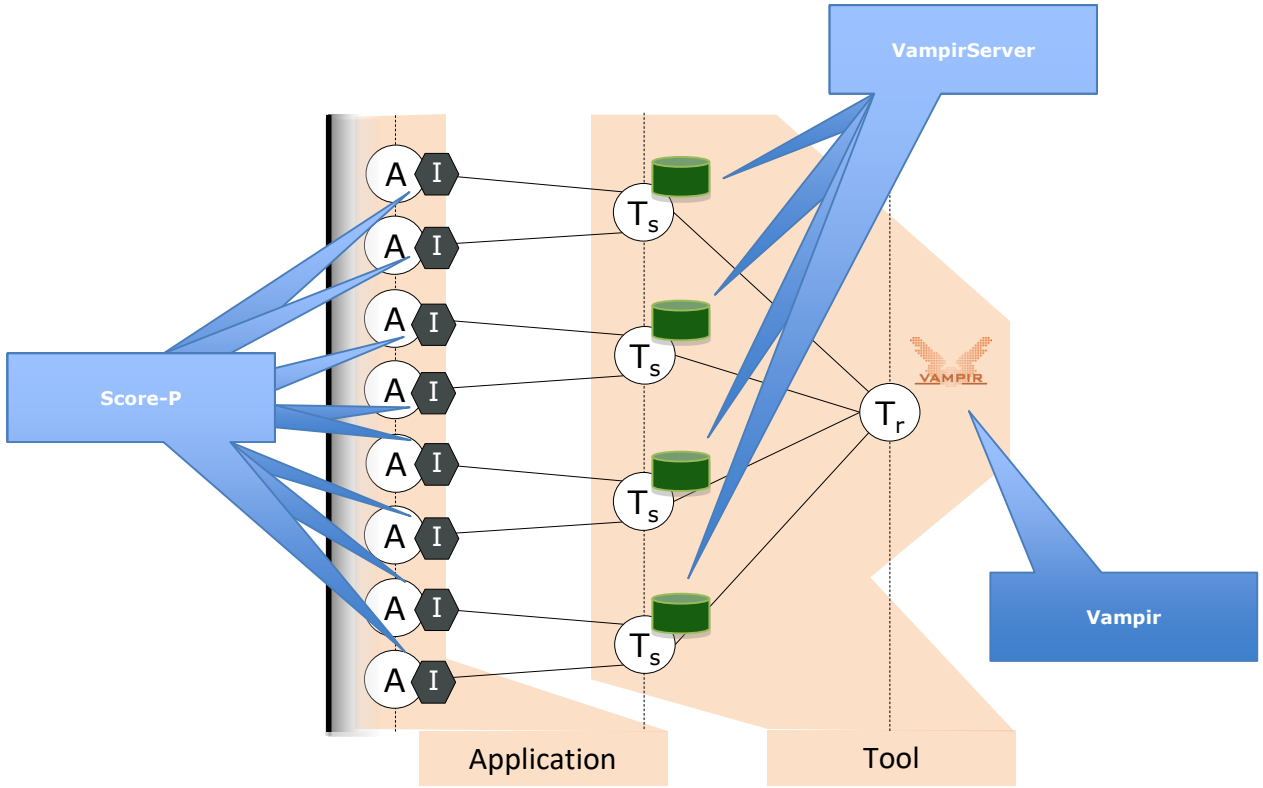
- *Tracing of Multi-Threaded Java Applications in Score-P Using JVMTI and User Instrumentation*
  Frenzel, J., Feldhoff, K., Jäkel, R. & Müller-Pfefferkorn, R., 2017
- *Advanced Python Performance Monitoring with Score-P*
  Gocht A., Schöne R., Frenzel J., 2021
- *Bridging between Data Science and Performance Analysis: Tracing of Jupyter Notebooks.*
  Werner, E., Manjunath, L., Frenzel, J., & Torge, S., 2021
- *I/O Recording and Workflow Analysis with Score-P and Vampir*
  Bill Williams, Scalable Tools Workshop 2019

➢ Single process only

➢ Multiple "trace merger" written over the past decade

Related Work

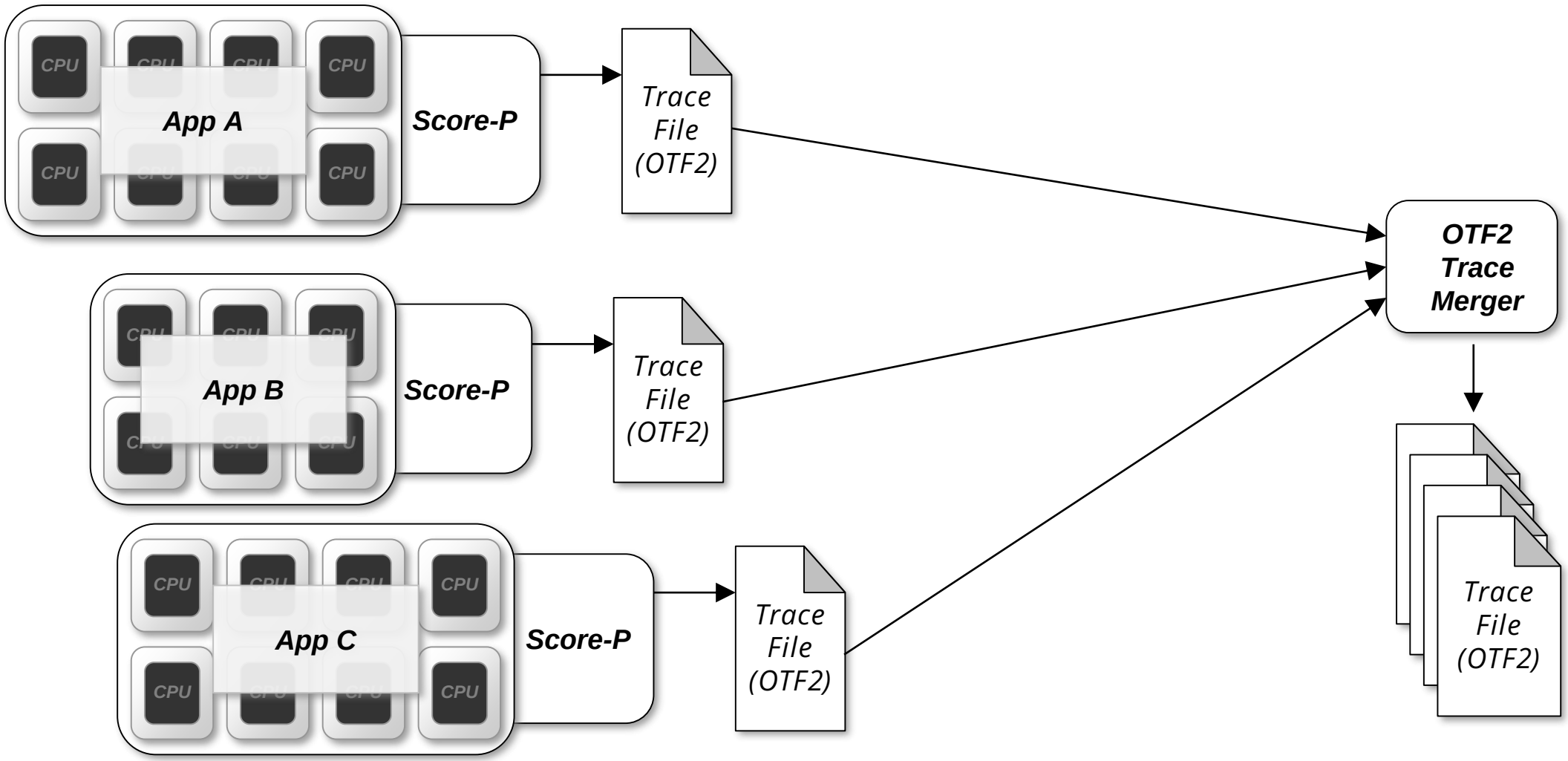
- *Online Performance Analysis with the Vampir Tool Set*
  M. Weber, J. Ziegenbalg, B. Wesarg, 2017
  - Trace data is kept at the lowest tool level
  - Sliced trace data is merged to have real time visualization at the tool root
  - In pause mode tool root can access raw trace data from the lowest level



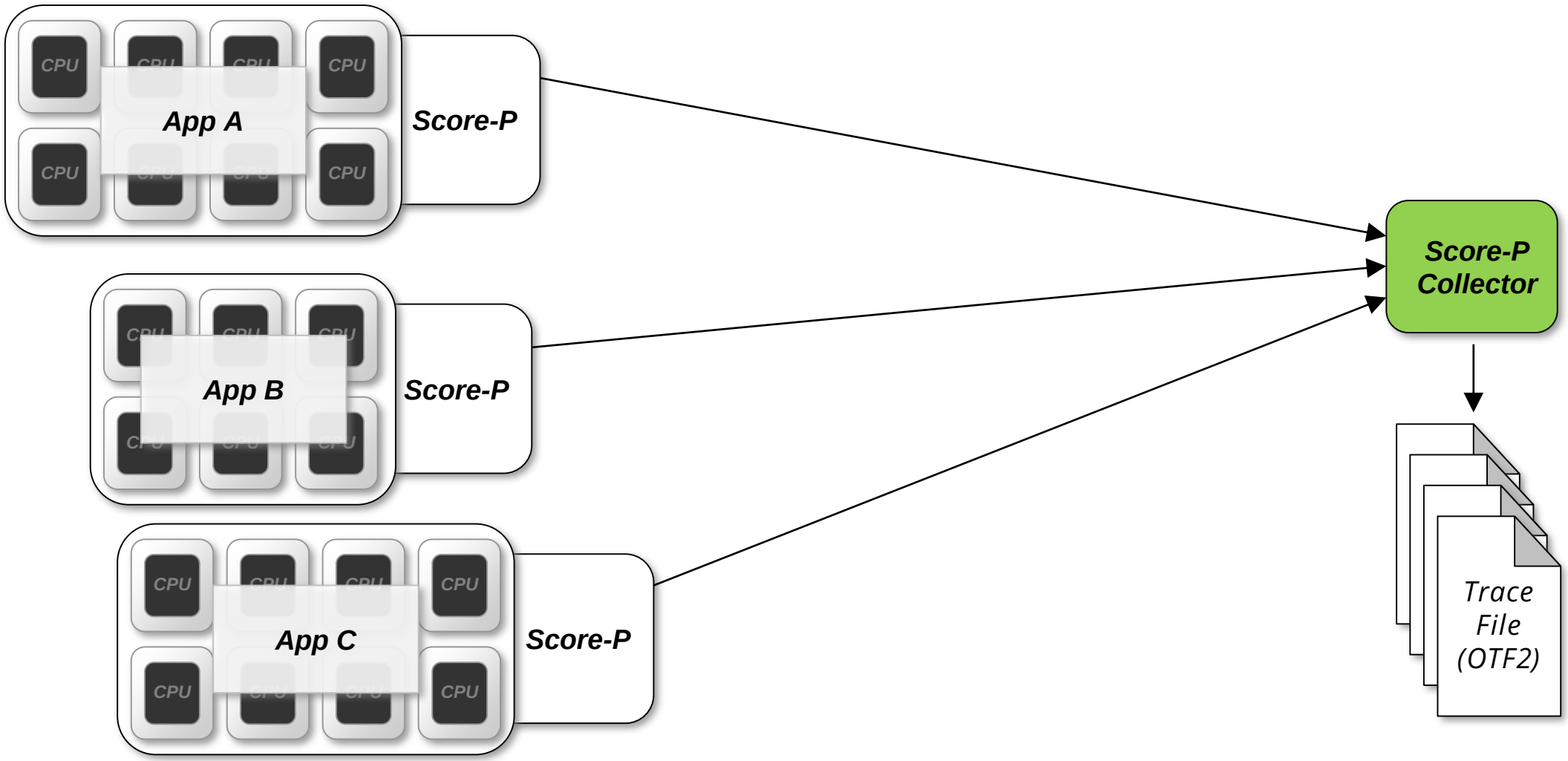
Reasons for slow adoption

- Multi-process applications which are not using MPI to coordinate
- Distributed application which do not share a file system
- Non-collective start of applications

Current state



The Idea


CPU
App A
Score-P
App B
Score-P
App C
Score-P
Score-P
Collector
Trace
File
(OTF2)

Requirements and limitations

- Performance data (and metadata) is held in memory for as long as possible and is only written at the end of the measurement
- Distributed across untrusted networks
- Re-use as much code from Score-P as possible in the collector

Client/server connection

- Using NanoMG-NG for client/server communication
  - C/C++
  - https://github.com/nanomsg/nng
  - With TLS support (https://github.com/Mbed-TLS/mbedtls)

Handling metadata (aka definition unification)

- Serialized in-memory format for definitions (virtual pointers)
- MPI/SHMEM: Embedded hypercube to unify definitions at finalization
- Component could be fully re-used by collector

Handling of trace event data

- Trace library (OTF2) implements chunked buffers which are flushed to disk (same memory and file representation)
- References to definitions are mapped at reading time to their globals
- Flushing is controlled by callbacks

➢ Chunks can be transferred as is to the collector and be written to disk

Handling of profile data

- Call-path profiles mainly used to score instrumentation overhead and to estimate trace buffer sizes to avoid intermediate buffer flushes
- Ignored for now
- Overhead scoring can still be done per single processes

Use cases

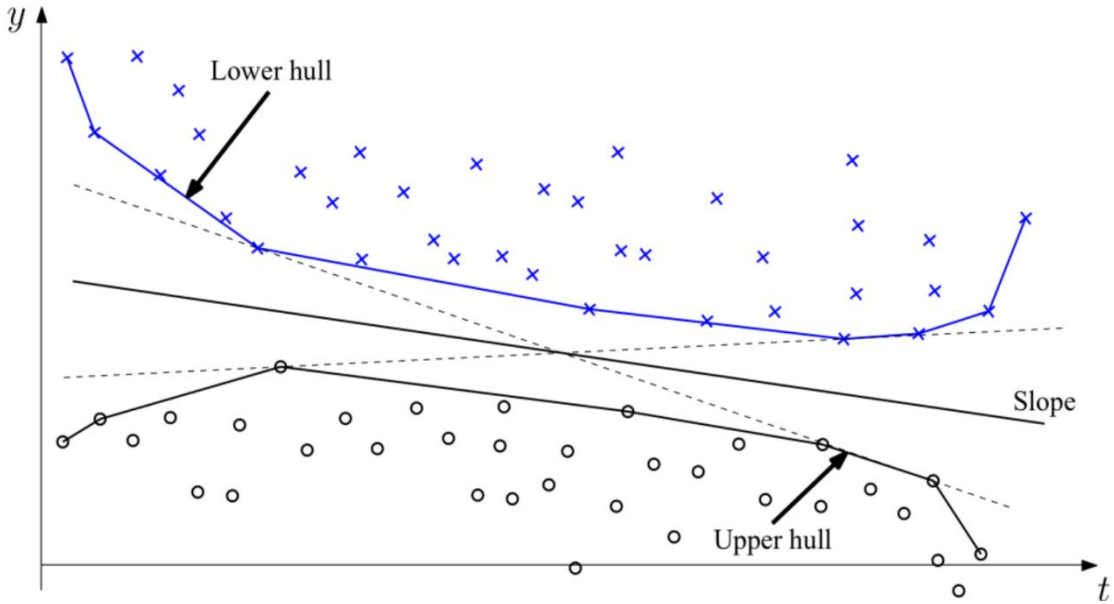
- Distributed Java/AI frameworks

- JupyterHub integration
  - Collector runs on local machine
  - Jupyter notebooks execute cells inside HPC jobs
  - Performance data is accessible from inside the notebook

- Embedded development (ARM, RISC-V, ...)

Outlook

- Foundation for distributed non-HPC frameworks
- Scalability
- Clock Synchronization
- MPI/SHMEM applications as clients
- Coordinator mode for MPMD style applications or clock synchronization



Continuous Clock Synchronization for Accurate Performance Analysis, J. Ziegenbalg, Scalable Tools Workshop 2017

On a different note: Vampir and JSON

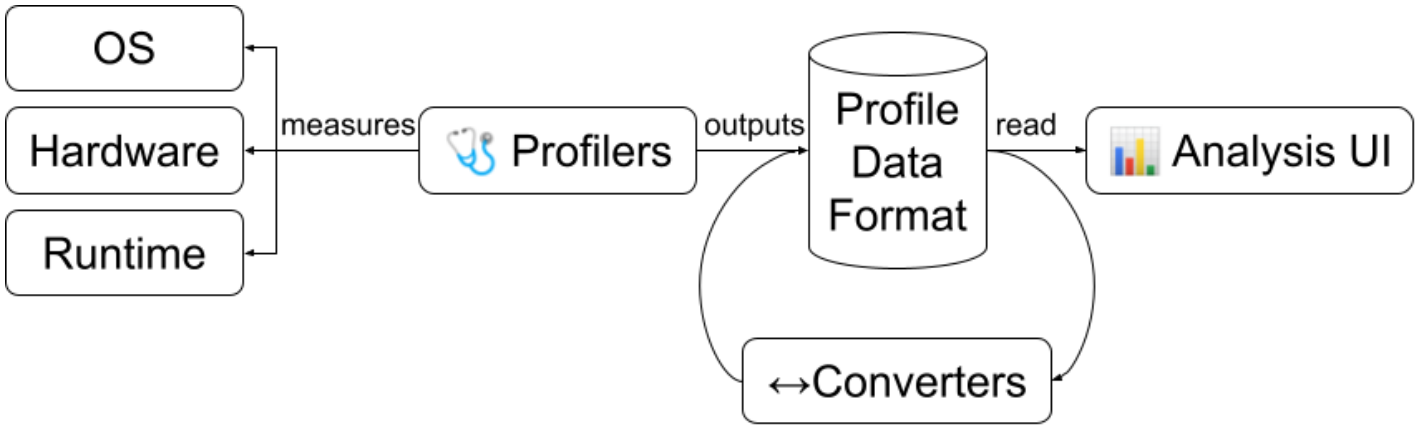
- "No" memory limit
  - — You shall not put the function name into every event
  - — You shall not use "combined" events (or at least sort them by the begin timestamp)
  - — You shall allow to read efficiently only a subset of the locations
- Allow to load JSON traces via URL

Profilerpedia

- "Profilerpedia is a catalog of Profilers, Profiler Data Formats they output, the UIs that can analyse/visualise those data formats, and data format converters."

- https://profilerpedia.markhansen.co.nz/

- 229 Profilers
- 160 Data Formats
- 129 Converters
- 147 Analysis UIs



Profilerpedia: HPC

Profiler platform: HPC

EZTrace

Profiler platforms: HPC

maqao lprof

Profiler platforms: HPC

Open|SpeedShop

Profiler platforms: HPC

Score-P

Profiler platforms: C C++ Fortran CUDA HIP Python MPI HPC