TECHNISCHE UNIVERSITÄT DRESDEN

Dresden Database Research Group

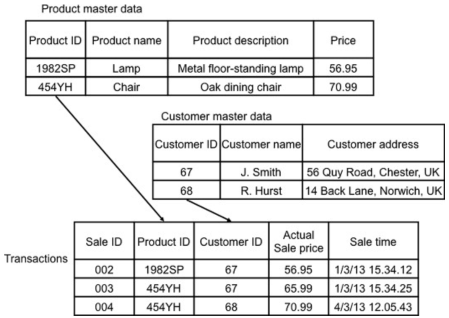
ScaleUp, ScaleOut, ….: Scale-"Flex" for the Next-Generation Database Engines?

Wolfgang Lehner & Alexander Krause

University of Waterloo – Lecture Series on "Disaggregated and Heterogeneous Computing Platform for Graph Processing"

Everything is Data → Variety is King!

Transactional Data:
Manual-curated business data
(Excel Sheets, Databases)



Product master data

Product ID	Product name	Product description	Price
1982SP	Lamp	Metal floor-standing lamp	56.95
454YH	Chair	Oak dining chair	70.99

Customer master data

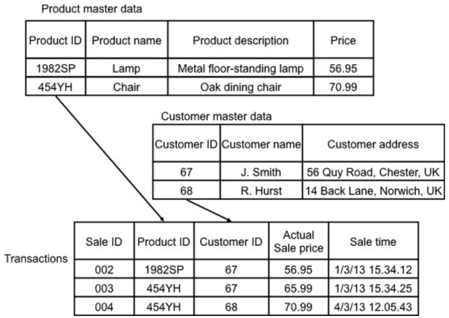
Customer ID	Customer name	Customer address
67	J. Smith	56 Quy Road, Chester, UK
68	R. Hurst	14 Back Lane, Norwich, UK

Transactions

Sale ID	Product ID	Customer ID	Actual Sale price	Sale time
002	1982SP	67	56.95	1/3/13 15.34.12
003	454YH	67	65.99	1/3/13 15.34.25
004	454YH	68	70.99	4/3/13 12.05.43

Everything is Data → Variety is King!

Transactional Data:
Manual-curated business data
(Excel Sheets, Databases)



Product master data

Product ID	Product name	Product description	Price
1982SP	Lamp	Metal floor-standing lamp	56.95
454YH	Chair	Oak dining chair	70.99

Customer master data

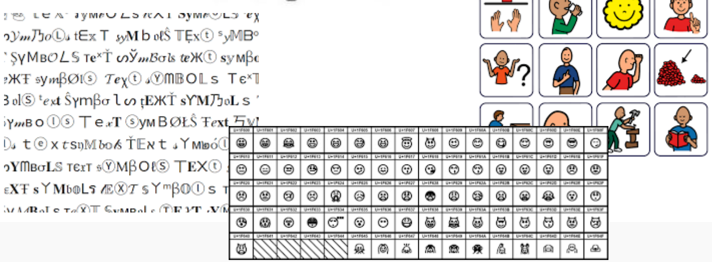
Customer ID	Customer name	Customer address
67	J. Smith	56 Quy Road, Chester, UK
68	R. Hurst	14 Back Lane, Norwich, UK

Transactions

Sale ID	Product ID	Customer ID	Actual Sale price	Sale time
002	1982SP	67	56.95	1/3/13 15.34.12
003	454YH	67	65.99	1/3/13 15.34.25
004	454YH	68	70.99	4/3/13 12.05.43

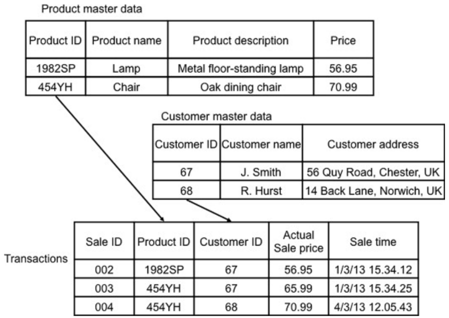
Machine Generated Data:
"Smart devices" generate
constantly data

Text and Image Data:
System and user
generated data



Everything is Data → Variety is King!

Transactional Data:
Manual-curated business data
(Excel Sheets, Databases)



Product master data

Product ID	Product name	Product description	Price
1982SP	Lamp	Metal floor-standing lamp	56.95
454YH	Chair	Oak dining chair	70.99

Customer master data

Customer ID	Customer name	Customer address
67	J. Smith	56 Quy Road, Chester, UK
68	R. Hurst	14 Back Lane, Norwich, UK

Transactions

Sale ID	Product ID	Customer ID	Actual Sale price	Sale time
002	1982SP	67	56.95	1/3/13 15.34.12
003	454YH	67	65.99	1/3/13 15.34.25
004	454YH	68	70.99	4/3/13 12.05.43

Data Pyramid

Data Replication

Machine Generated Data:
“Smart devices” generate
constantly data

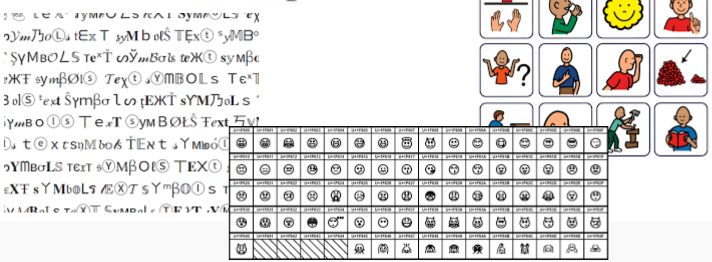
Lambda-Architectures

Data Mesh

Data Lakes

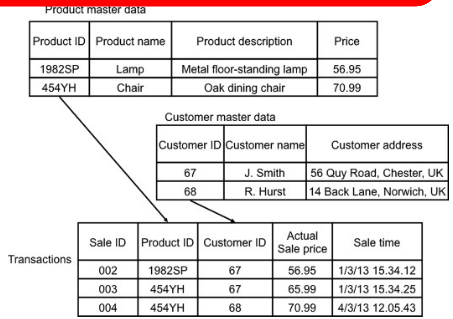
Open Table Formats

Text and Image Data:
System and user
generated data



Everything is Data → Variety is King!

Transactional Data:
Manual-curated business data
(Excel Sheets, Databases)



Product master data

Product ID	Product name	Product description	Price
1982SP	Lamp	Metal floor-standing lamp	56.95
454YH	Chair	Oak dining chair	70.99

Customer master data

Customer ID	Customer name	Customer address
67	J. Smith	56 Quy Road, Chester, UK
68	R. Hurst	14 Back Lane, Norwich, UK

Transactions

Sale ID	Product ID	Customer ID	Actual Sale price	Sale time
002	1982SP	67	56.95	1/3/13 15.34.12
003	454YH	67	65.99	1/3/13 15.34.25
004	454YH	68	70.99	4/3/13 12.05.43

Data Pyramid

Data Replication

Lambda-Architectures

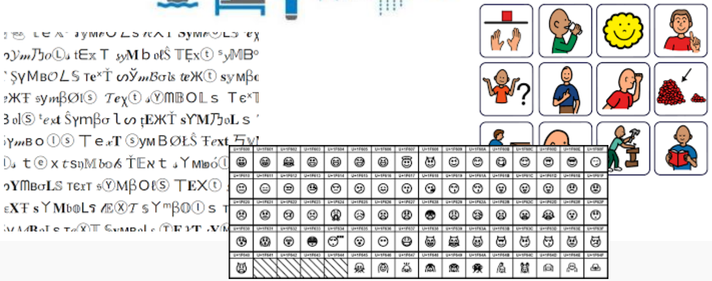
Data Mesh

Data Lakes

Open Table Formats

Machine Generated Data:
"Smart devices" generate
constantly data

Text and Image Data:
System and user
generated data



Plethora of Hardware → Variety is King!

CPU

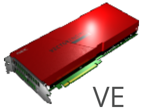
TPU

FPGA

ARM® big.LITTLE

SmartNIC

GPGPU

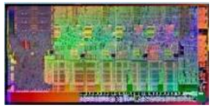


VE

ASIC

NVMe

RDMA, …



HBM

DRAM DIMM

PMEM DIMM

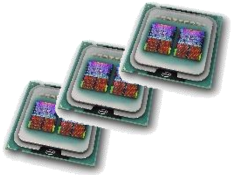
SSDs

…

pictures © by www.intel.com, www.nec.com, www.google.com

Plethora of Hardware → Variety is King!

CPU



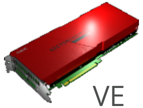
TPU

FPGA

ARM big.LITTLE

SmartNIC

GPGPU

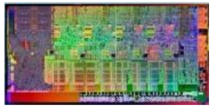


VE

ASIC

NVMe

RDMA, ...



HBM

DRAM DIMM

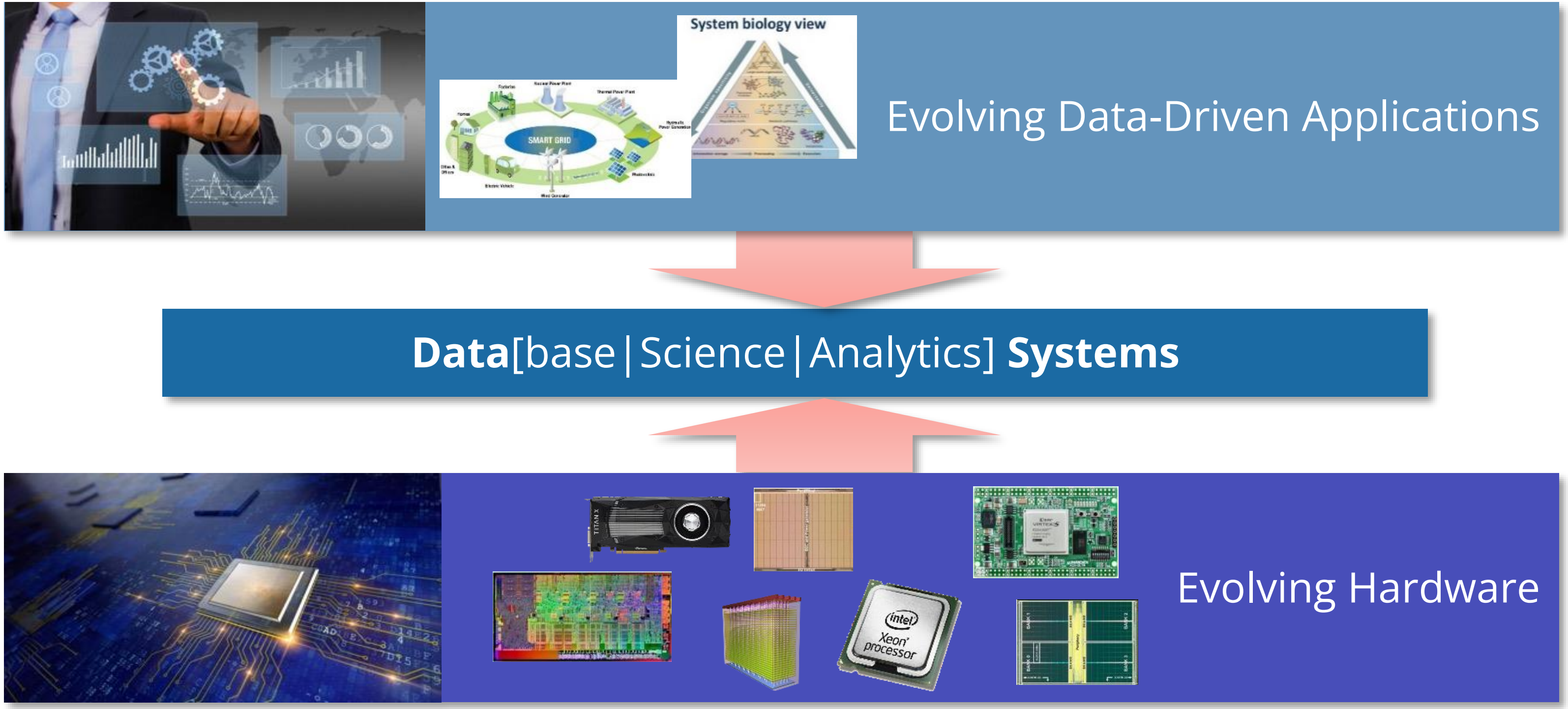
PMEM DIMM

SSDs

...

pictures © by www.intel.com, www.nec.com, www.google.com

Mission: Tackling the “Data Systems” Sandwich

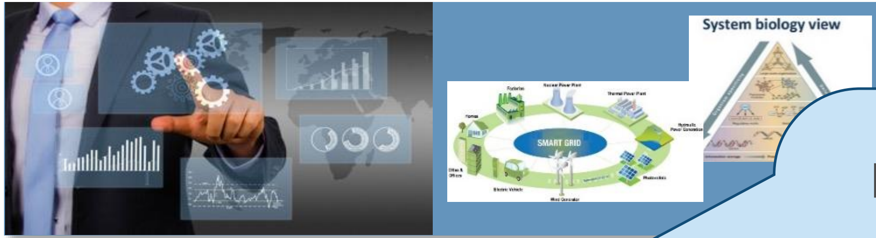


Evolving Data-Driven Applications

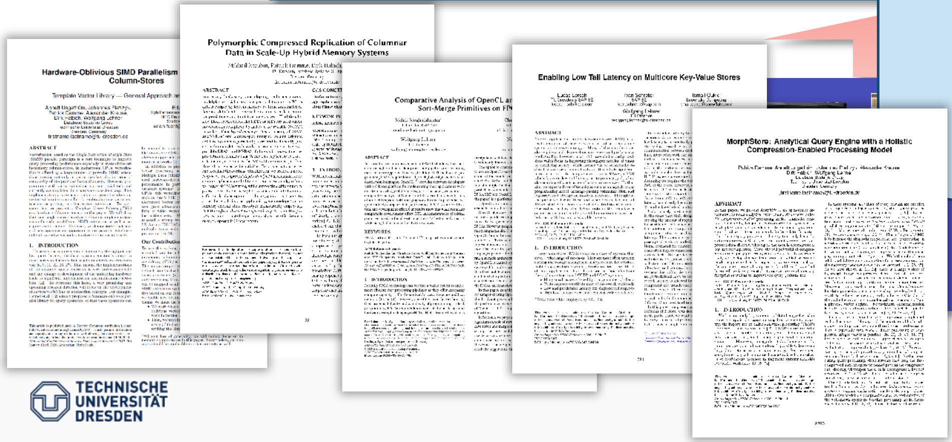
Data[base|Science|Analytics] **Systems**

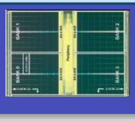
Evolving Hardware

Mission: Tackling the “Data Systems” Sandwich



Data[base|Scienc


Hardware-Oblivious SIMD Parallelism
Column-Stores
Polymorphic Compressed Replication of Columnar Data in Scale-Up Hybrid Memory Systems
Comparative Analysis of OpenCL ar
Sort-Merge Primitives on FP
Enabling Low Tail Latency on Multicore Key-Value Stores
MorphStore: Analytical Query Engine with a Holistic Compression-Enabled Processing Model



Key Question:

How to build a
- highly scalable,
- highly elastic,
- highly robust

Data[base|Science|Analytics] **System**?

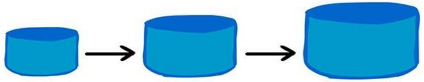
Design Space for Data Systems Architectures

Scale-Up (scale vertically)

- ➢ add resources to a single node in a system, typically CPUs or memory
- ➢ big iron – expensive.

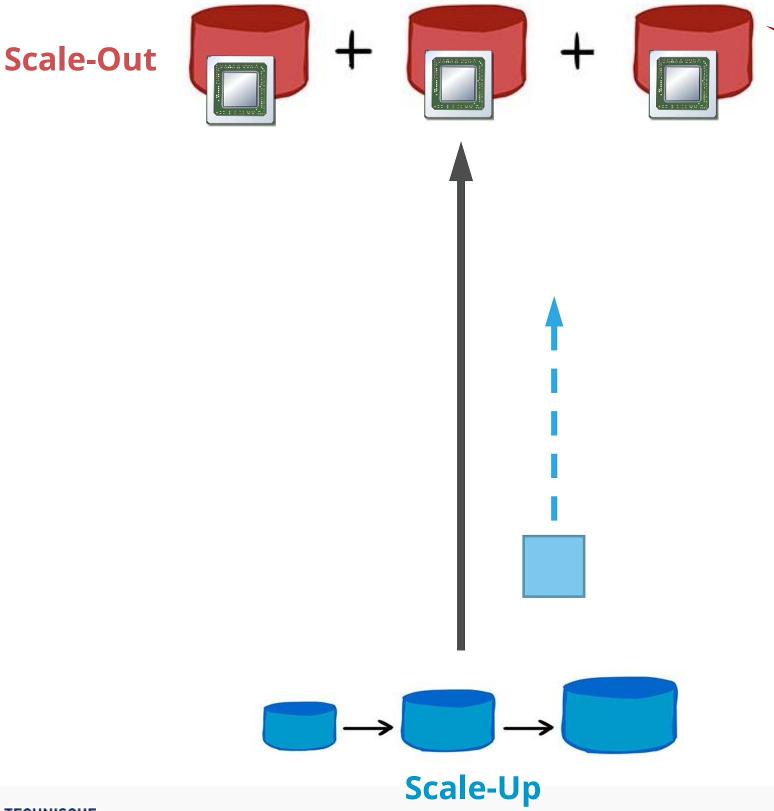
The traditional world of (transactional) DBMS

- single address space
- (mostly) homogenous processing units
- robust HW design



Scale-Up

Design Space for Data Systems Architectures



Scale-Out

Scale-Up

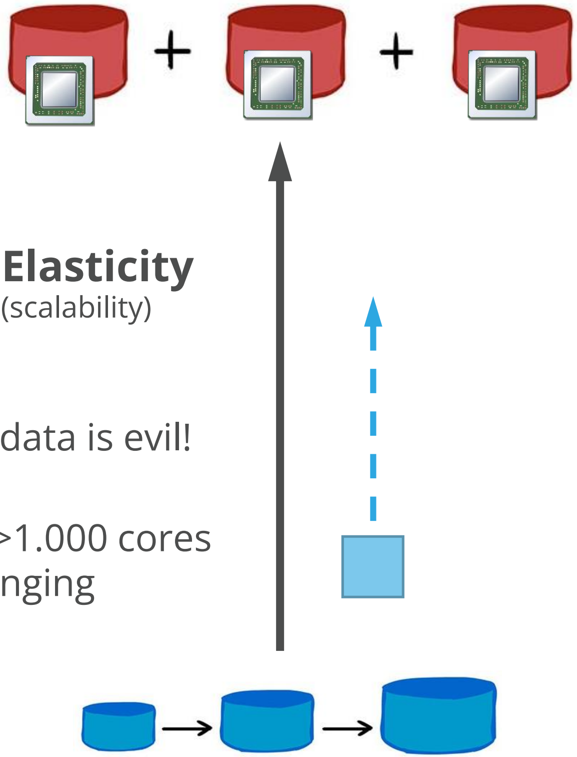
Scale-Out (scale horizontally)

- add more nodes to a system, such as adding a new computer to a distributed software application
- cluster of commodity hardware

The world of the "Sparks"/"Flinks" e.a.

- implement data lakes, …
- separation of compute/store

Design Space for Data Systems Architectures



⚠ moving data is evil!

⚠ taming >1.000 cores is challenging

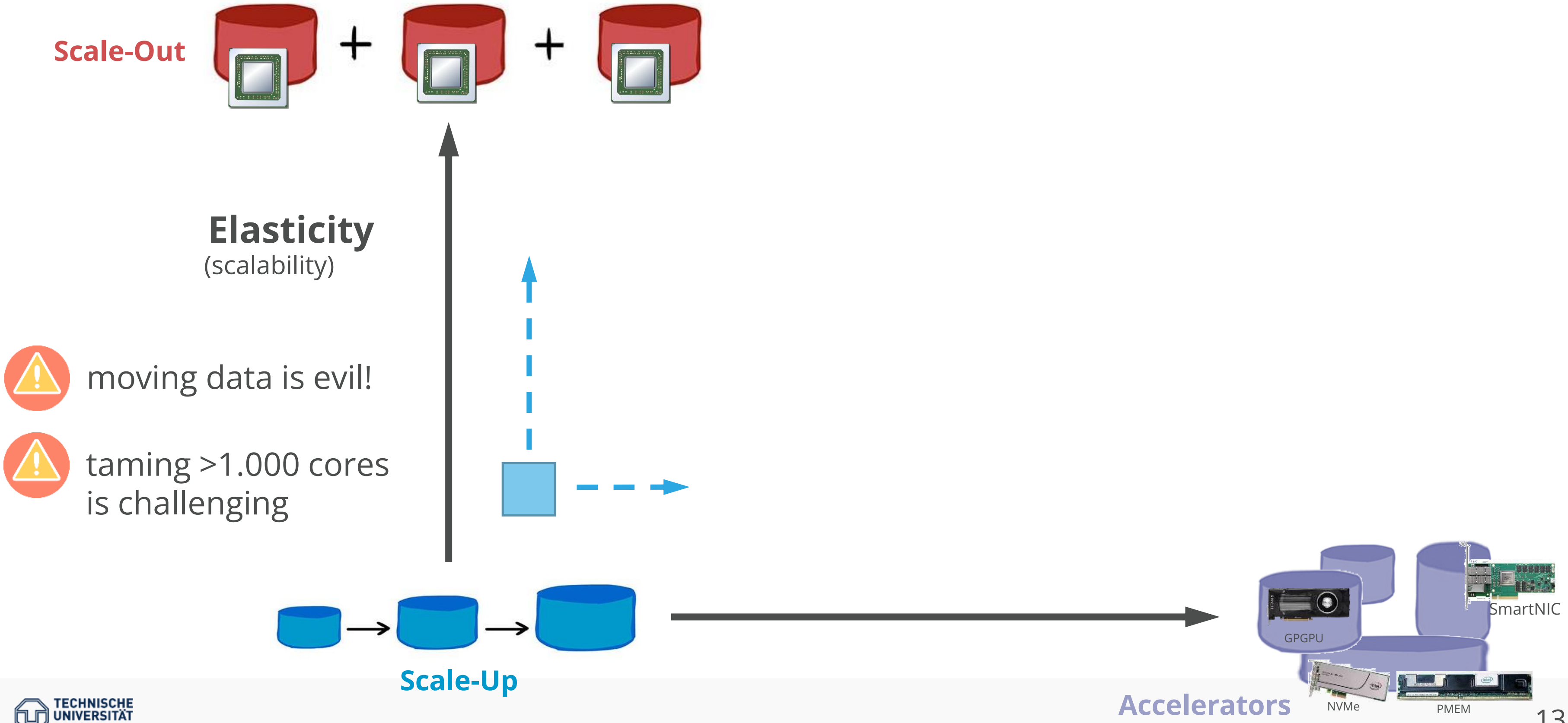
Scale-Out (scale horizontally)

- ➢ add more nodes to a system, such as adding a new computer to a distributed software application
- ➢ cluster of commodity hardware

The world of the "Sparks"/"Flinks" e.a.

- implement data lakes, …
- separation of compute/store

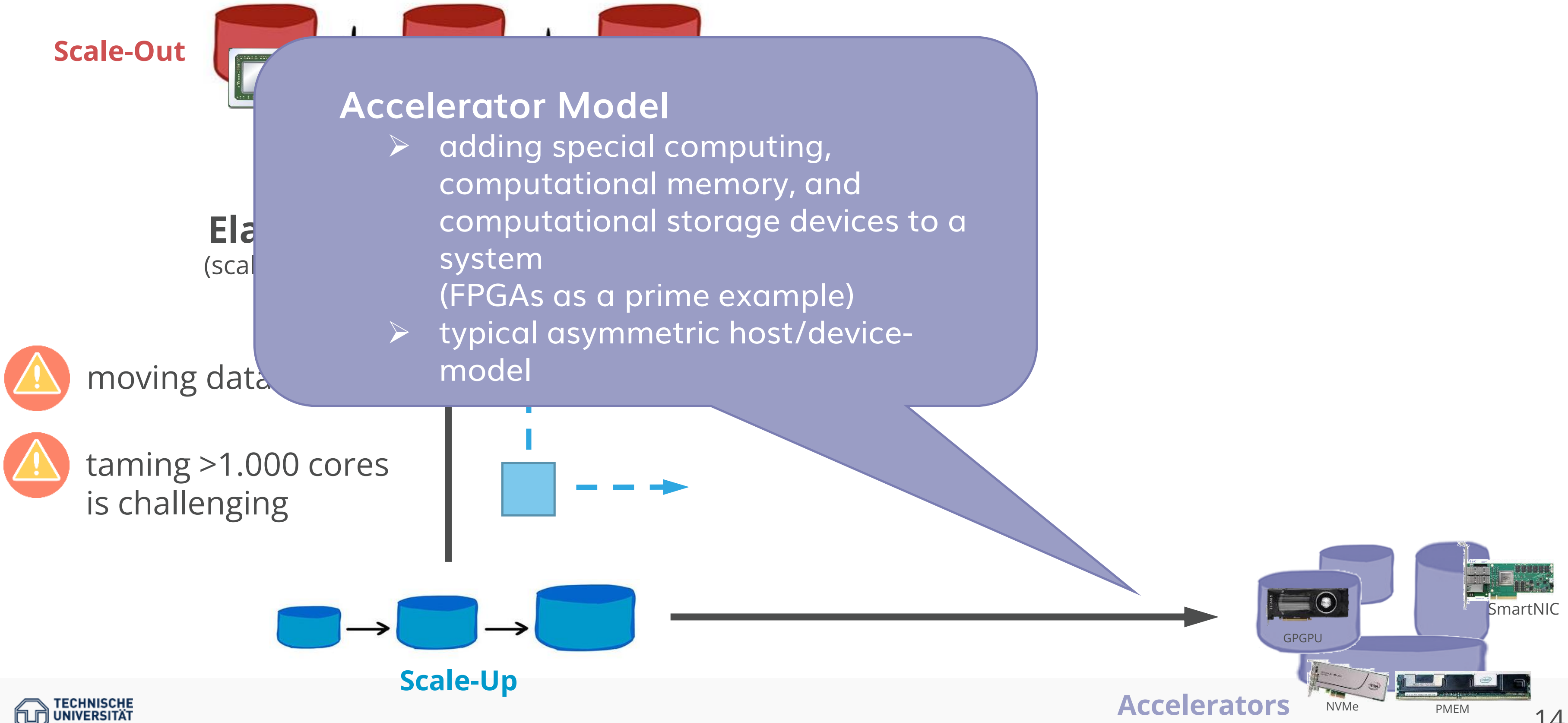
Design Space for Data Systems Architectures



⚠ moving data is evil!

⚠ taming >1.000 cores is challenging

Design Space for Data Systems Architectures



Accelerator Model

- adding special computing, computational memory, and computational storage devices to a system
  (FPGAs as a prime example)
- typical asymmetric host/device-model

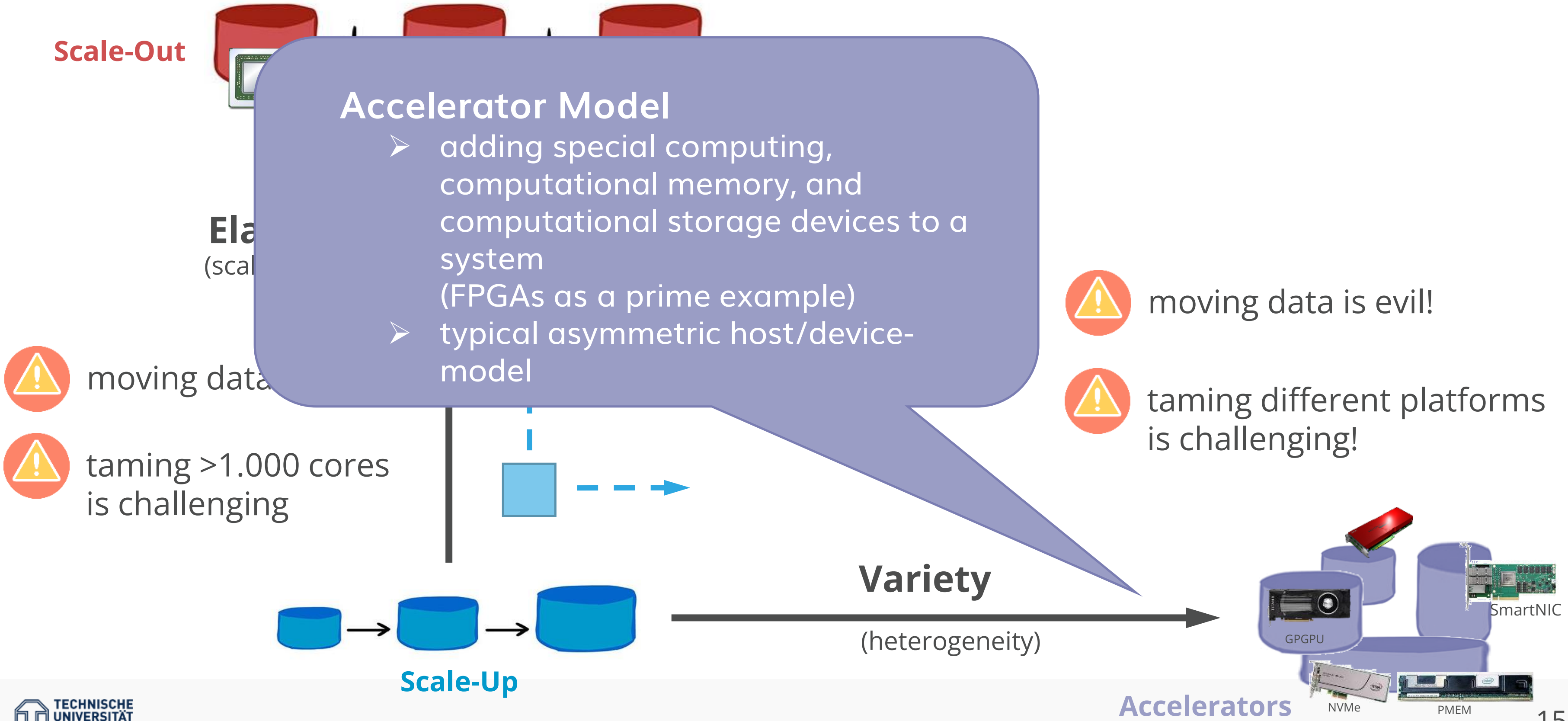
⚠ moving data

⚠ taming >1.000 cores is challenging

Design Space for Data Systems Architectures

Accelerator Model

- adding special computing, computational memory, and computational storage devices to a system
  (FPGAs as a prime example)
- typical asymmetric host/device-model



Scale-Out

Ela
(scal

⚠ moving data

⚠ taming >1.000 cores is challenging

⚠ moving data is evil!

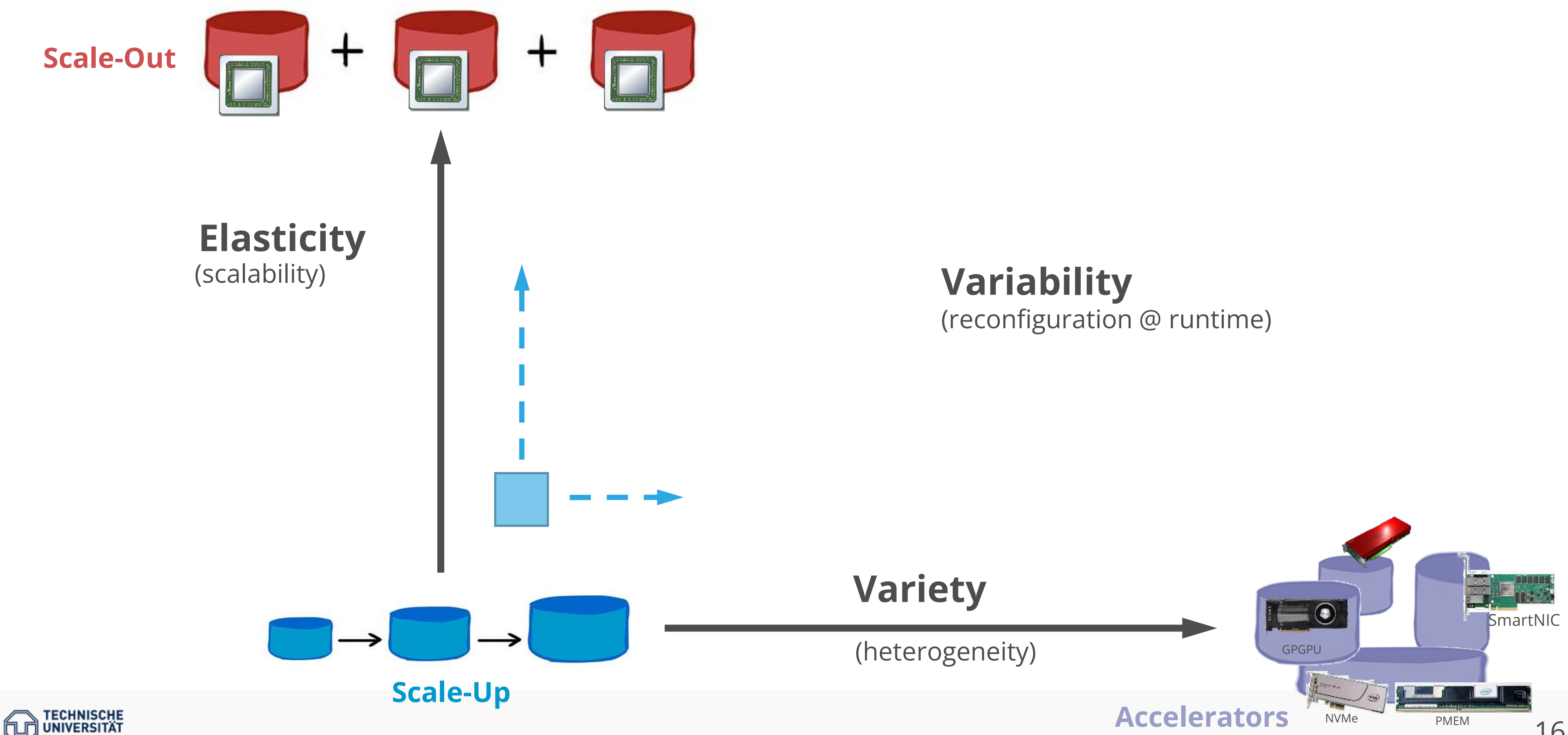
⚠ taming different platforms is challenging!

Variety
(heterogeneity)

Scale-Up

Accelerators

Design Space for Data Systems Architectures



Scale-Out

Elasticity
(scalability)

Variability
(reconfiguration @ runtime)

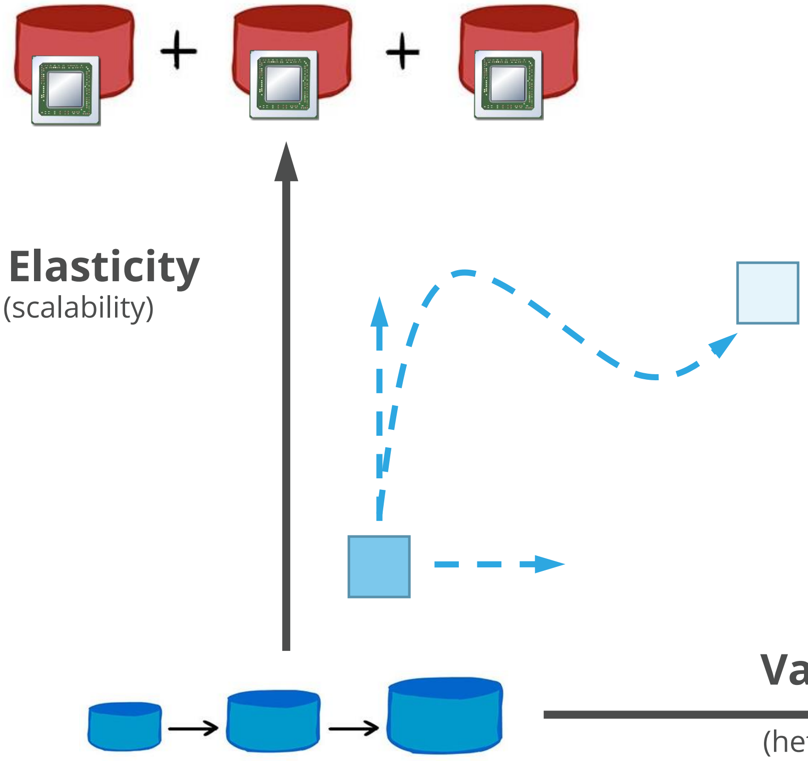
Variety

(heterogeneity)

Scale-Up

Accelerators

Design Space for Data Systems Architectures



Scale-Out

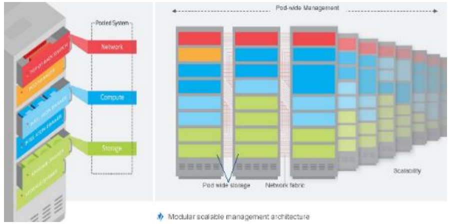
Elasticity
(scalability)

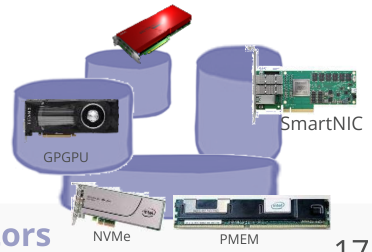
Variability
(reconfiguration @ runtime)

Variety
(heterogeneity)

Scale-Up

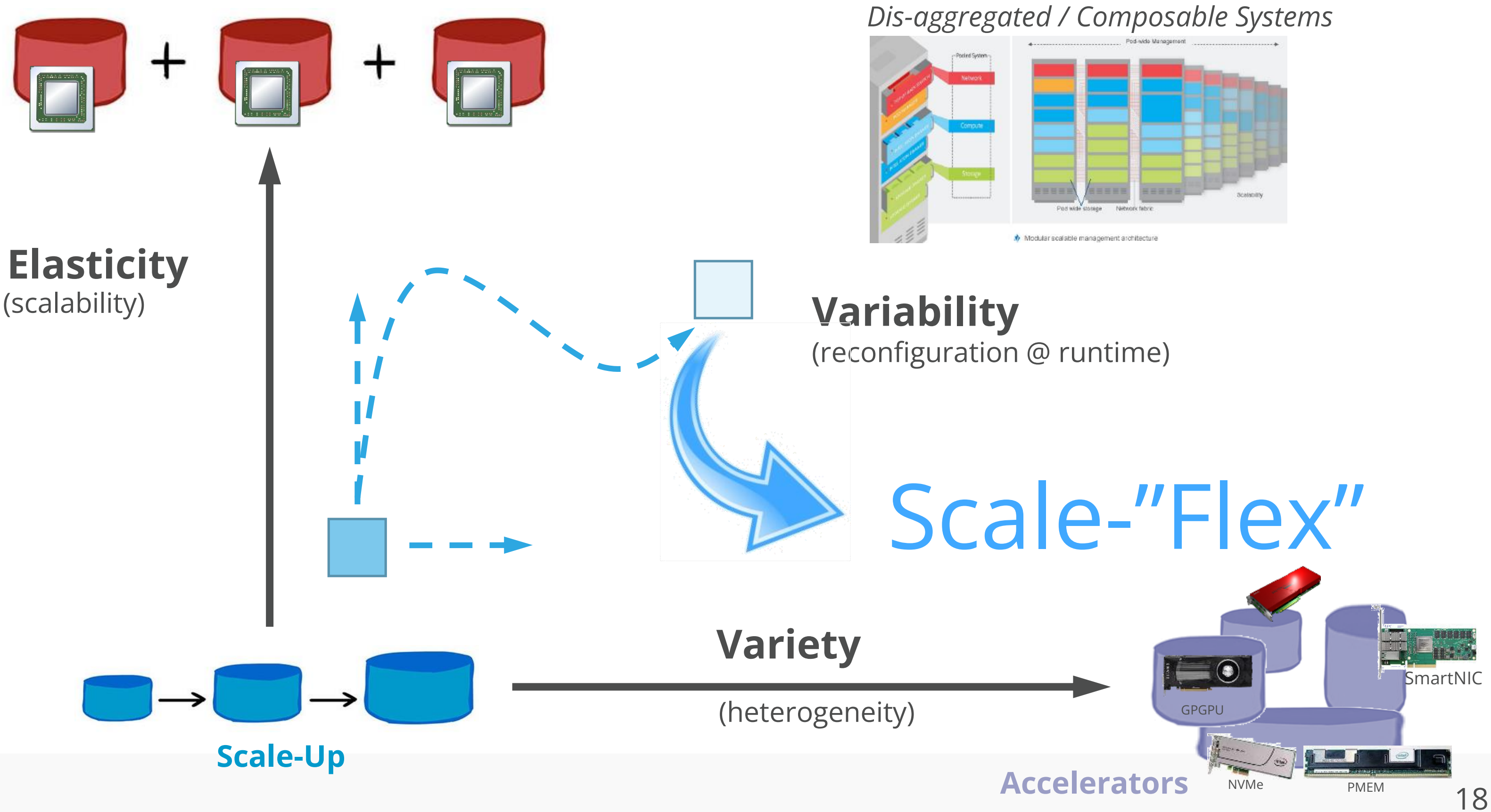
Dis-aggregated / Composable Systems





Accelerators

Design Space for Data Systems Architectures



Scale-Out

Elasticity
(scalability)

Dis-aggregated / Composable Systems

Variability
(reconfiguration @ runtime)

Scale-"Flex"

Variety
(heterogeneity)

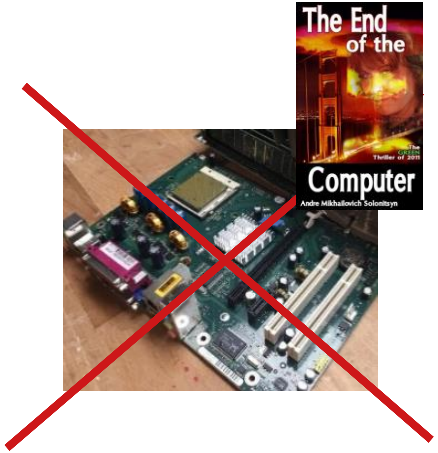
Scale-Up

Accelerators

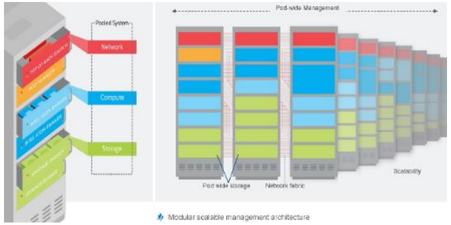
Tackling Elasticity & Variety: Scale-"Flex"

IN USE

AT REST



Composable Systems

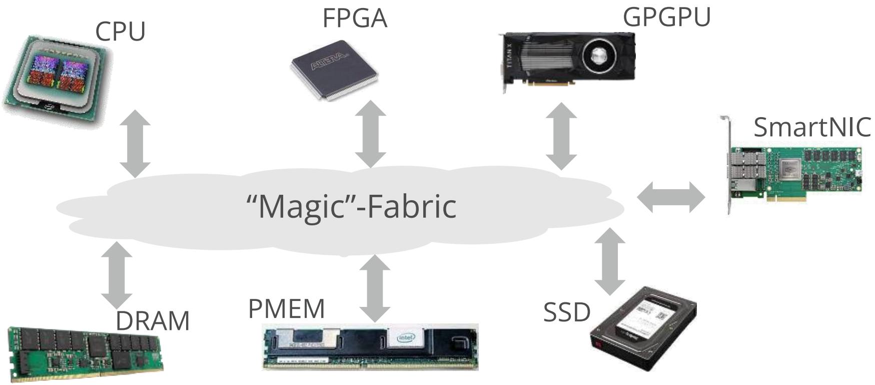


Fully disaggregated compute and memory/storage resources

Tackling Elasticity & Variety : Scale-"Flex"

IN USE

AT REST

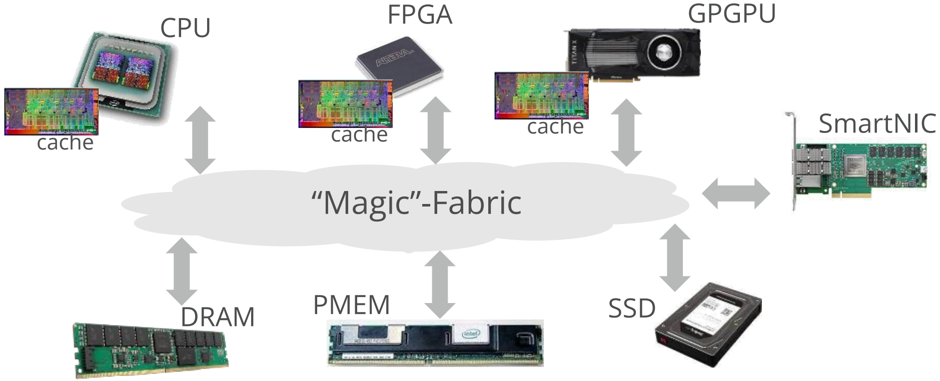


Pool of resources that can be made available on demand

Tackling Elasticity & Variety: Scale-"Flex"

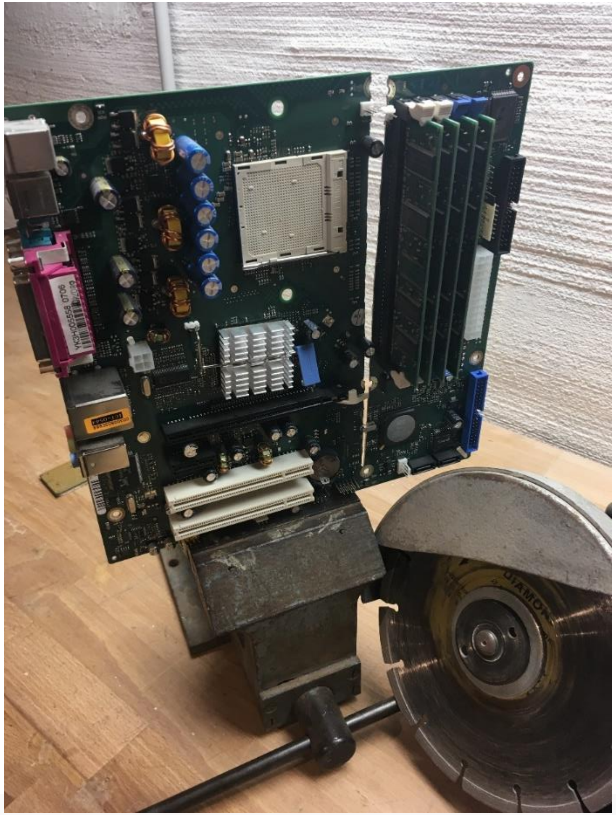
IN USE

AT REST



Pool of resources that can be made available on demand with common memory address space

My First Try: ...not so successful!



CXL
Compute Express Link

Consortium initiated by Intel e.a.

Open standard for communication between

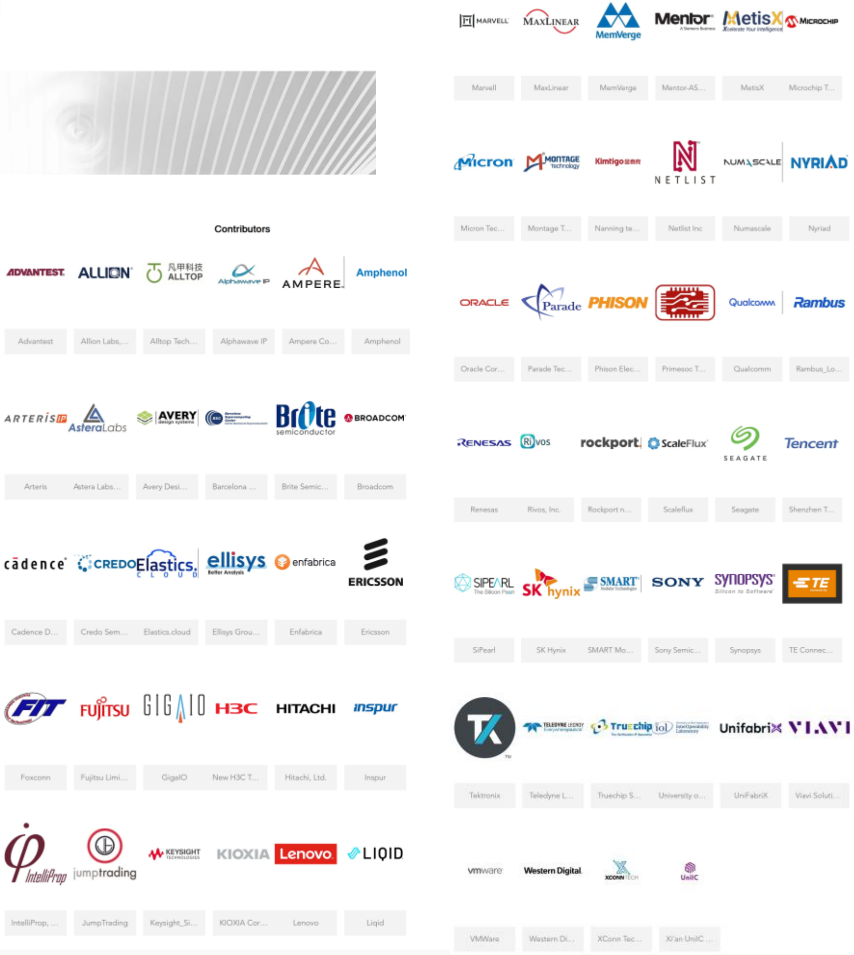
- CPU-to-device
- CPU-to-memory

Specification

- 1.0 (March 2019)
- 1.1 (June 2019)
- [2.0 (Nov 2020)]
- 3.0 (Aug 2022)

Board of Directors

Alibaba | Advanced Micro D... | Arm | Cisco | Dell
Google | HPE | Huawei L | IBM | Intel Corporation
Meta | Microsoft | NVIDIA | Rambus | Samsung


Contributors

Advantest | Allion Labs,... | Alltop Tech... | Alphawave IP | Ampere Co... | Amphenol
Arteris | Astera Labs... | Avery Desi... | Barcelona ... | Brite Semic... | Broadcom
Cadence D... | Credo Sem... | Elastics.cloud | Ellisys Grou... | Enfabrica | Ericsson
Foxconn | Fujitsu Limi... | GigaIO | New H3C T... | Hitachi, Ltd. | Inspur
IntelliProp, ... | JumpTrading | Keysight_Si... | KIOXIA Cor... | Lenovo | Liqid

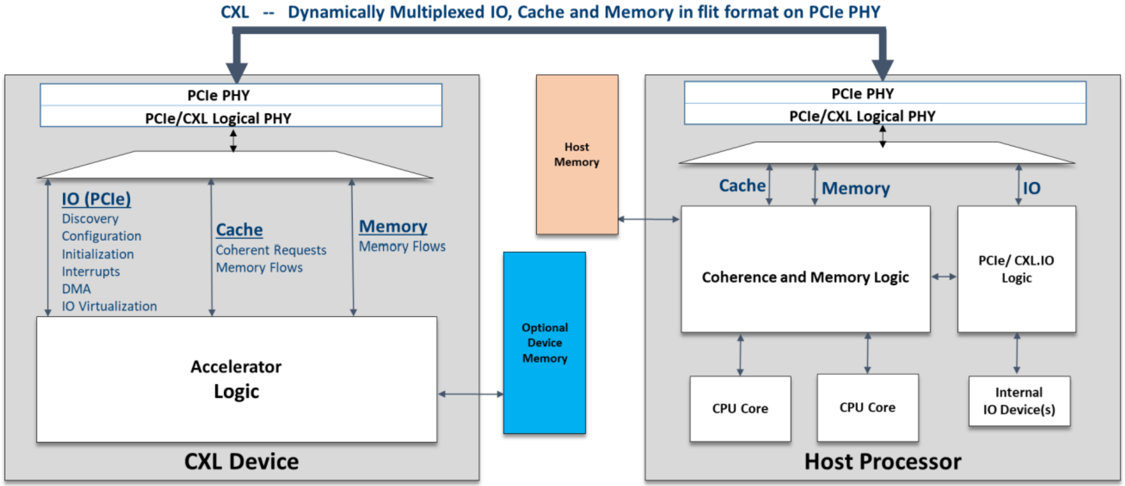
Marvell | MaxLinear | MemVerge | Mentor-AS... | MetisX | Microchip T...
Micron Tec... | Montage T... | Nanning te... | Netlist Inc | Numascale | Nyriad
Oracle Cor... | Parade Tec... | Phison Elec... | Primesoc T... | Qualcomm | Rambus_Lo...
Renesas | Rivos, Inc. | Rockport n... | Scaleflux | Seagate | Shenzhen T...
SiPearl | SK Hynix | SMART Mo... | Sony Semic... | Synopsys | TE Connec...
Tektronix | Teledyne L... | Truechip S... | University o... | UniFabriX | Viavi Soluti...
VMWare | Western Di... | XConn Tec... | Xi'an UnilC ...

3 core protocols
-> 3 common use cases

CXL Protocols

Principles

- maintains a unified, coherent memory space between the CPU (host processor) and any memory on any attached CXL device
- share resources and operate on the same memory region in order to reduce necessity of data-movement



CXL.IO

- PCIe based / discovery, register access, interrupts, initialization, I/O Virtualization, DMA

CXL.Cache

- supports device caching of host memory with host processor orchestrating the coherency management

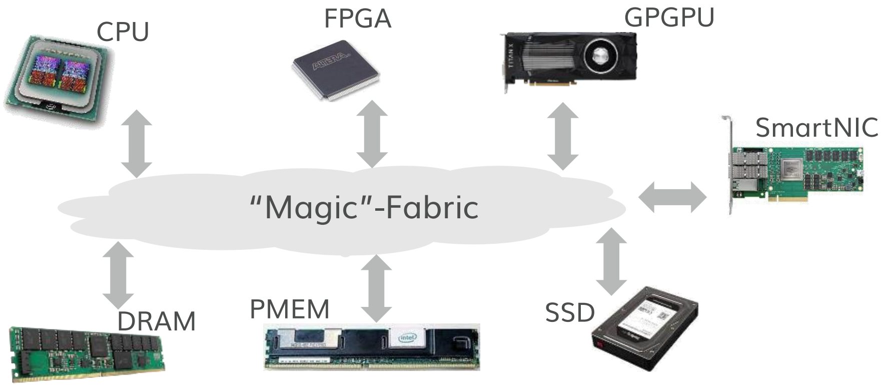
CXL.Memory

- memory access protocol, host manages (coherency) device attached memory similar to host memory

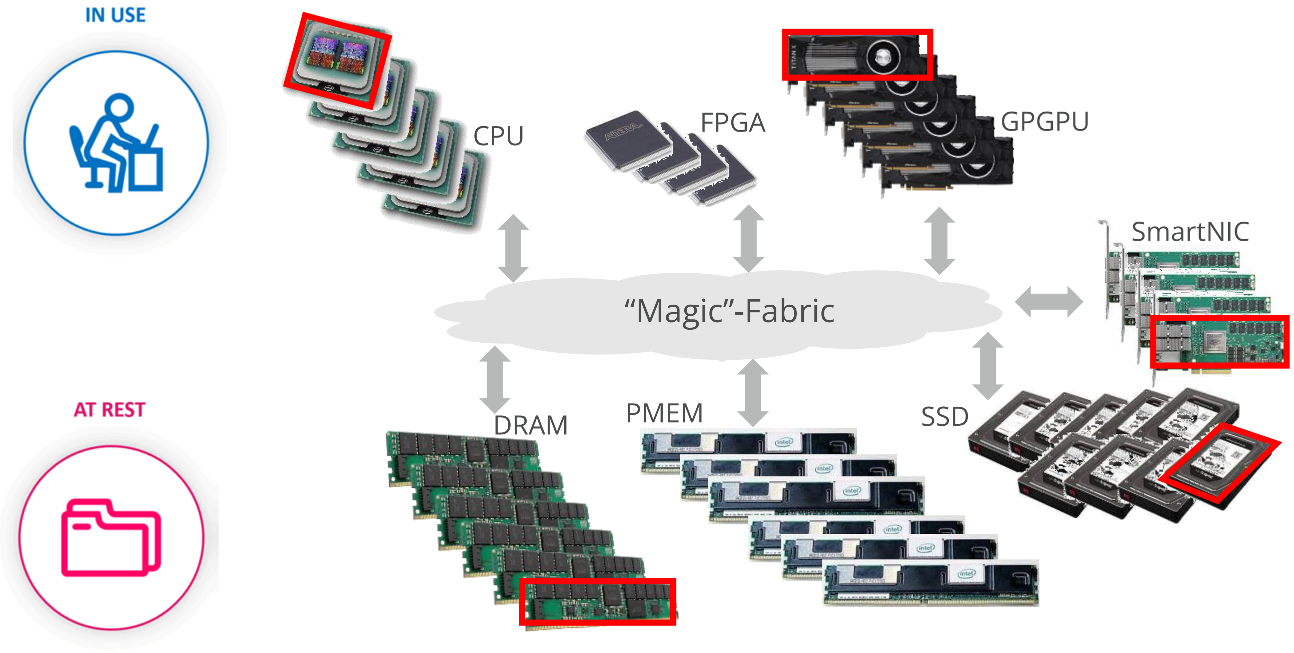
Recap: Tackling Elasticity & Variety : Scale-"Flex"

IN USE

AT REST

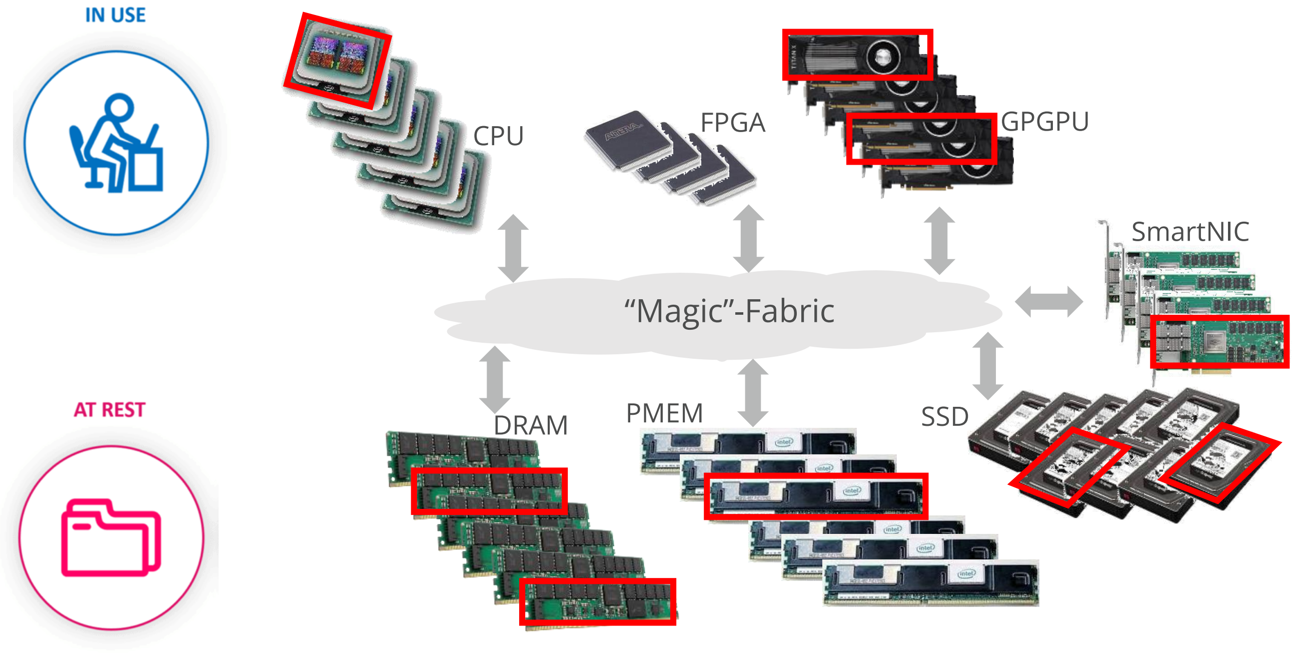


Scale-"Flex" as Breathing Infrastructure



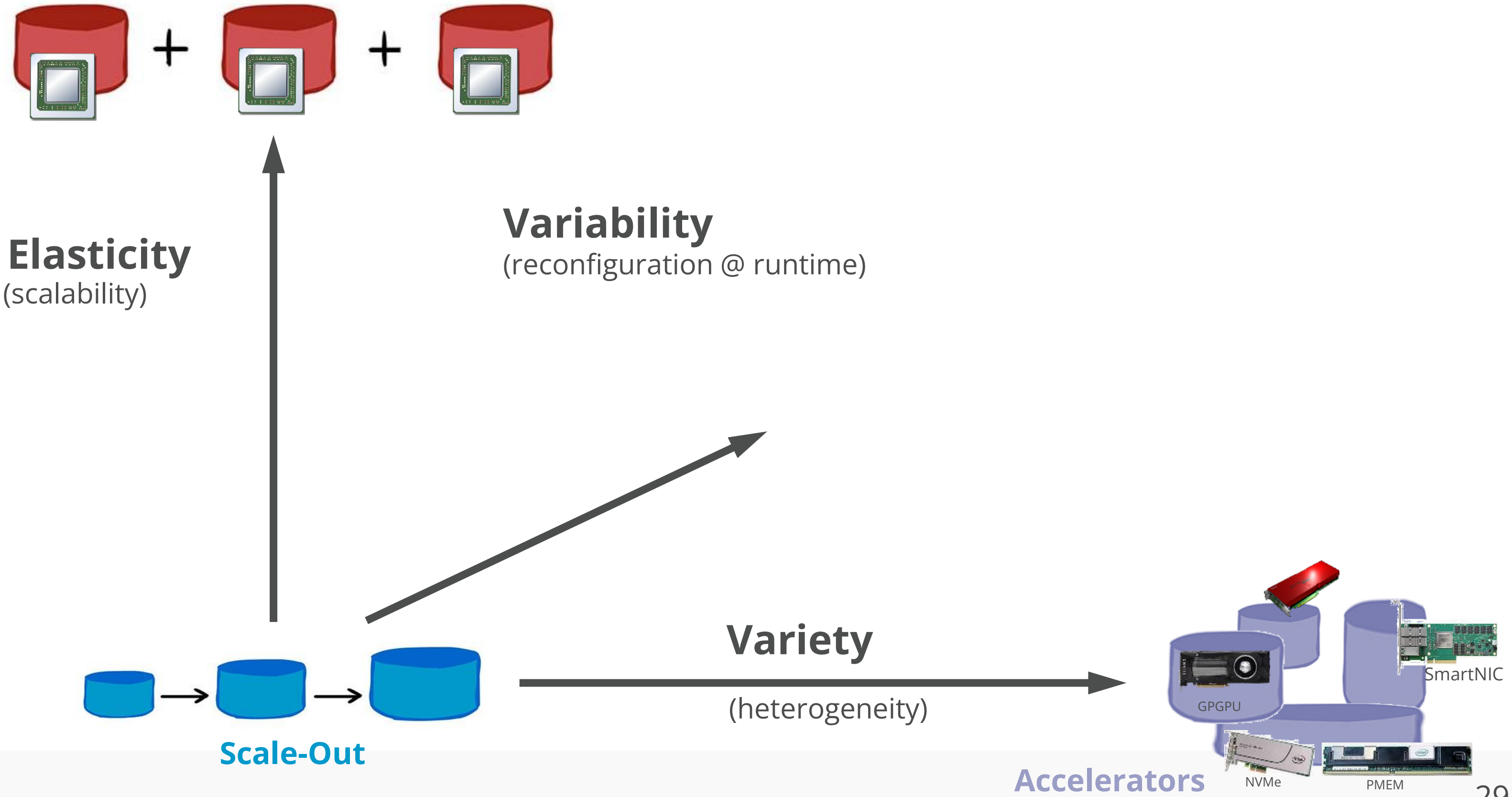
COMPOSITION of devices to form "a Virtual Machine" by **software**
→ eventually also by the **Data[base|Science|Analytics] System**

Scale-"Flex" as Breathing Infrastructure



RECONFIGURATION of CXL devices during runtime
→ eventually also by the **Data[base|Science|Analytics] System**

Design Space for Data Systems Architectures



Scale-Out

Elasticity
(scalability)

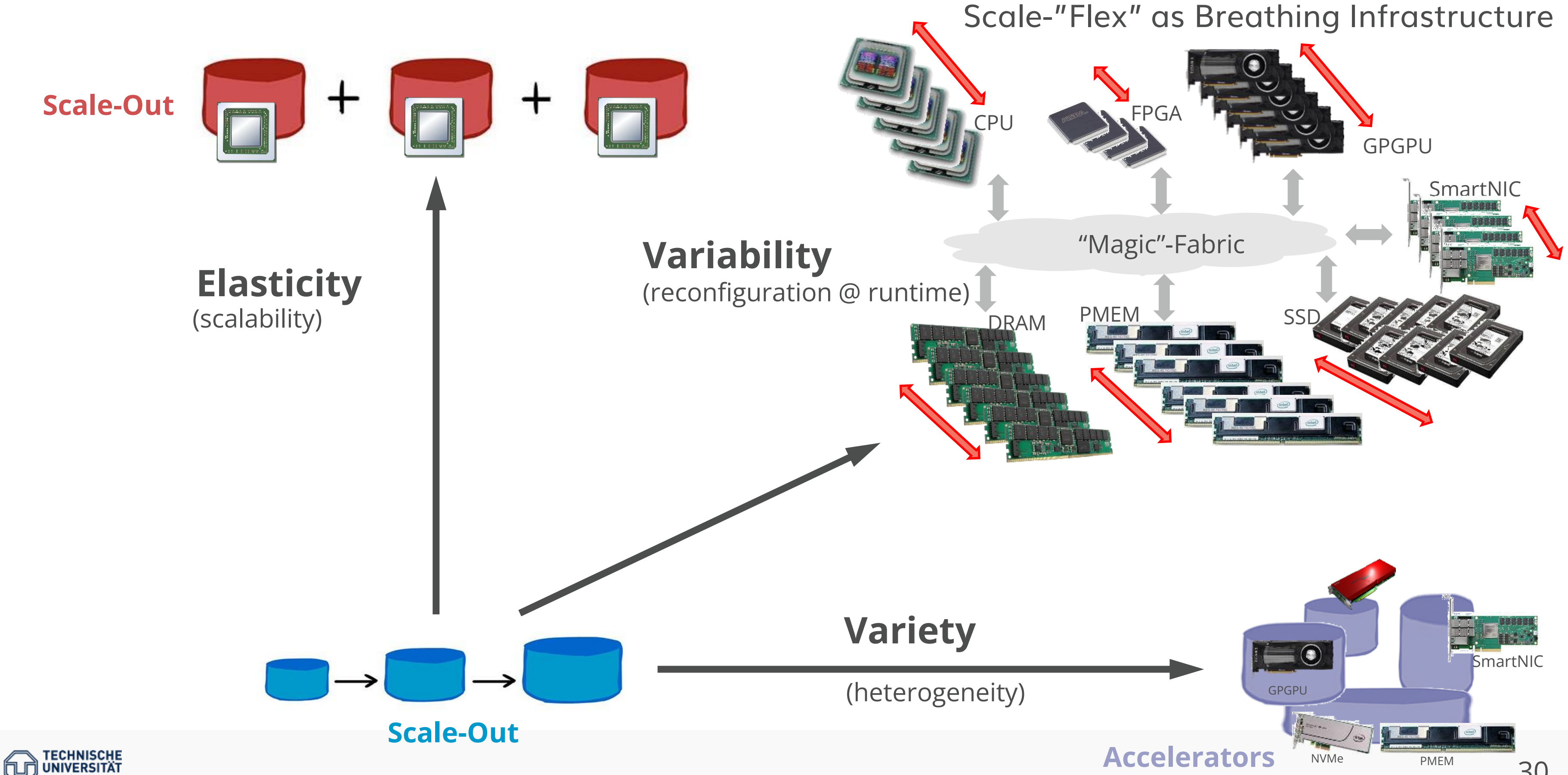
Variability
(reconfiguration @ runtime)

Variety
(heterogeneity)

Scale-Out

Accelerators

Design Space for Data Systems Architectures



Scale-"Flex" as Breathing Infrastructure

Scale-Out

Elasticity

(scalability)

Variability

(reconfiguration @ runtime)

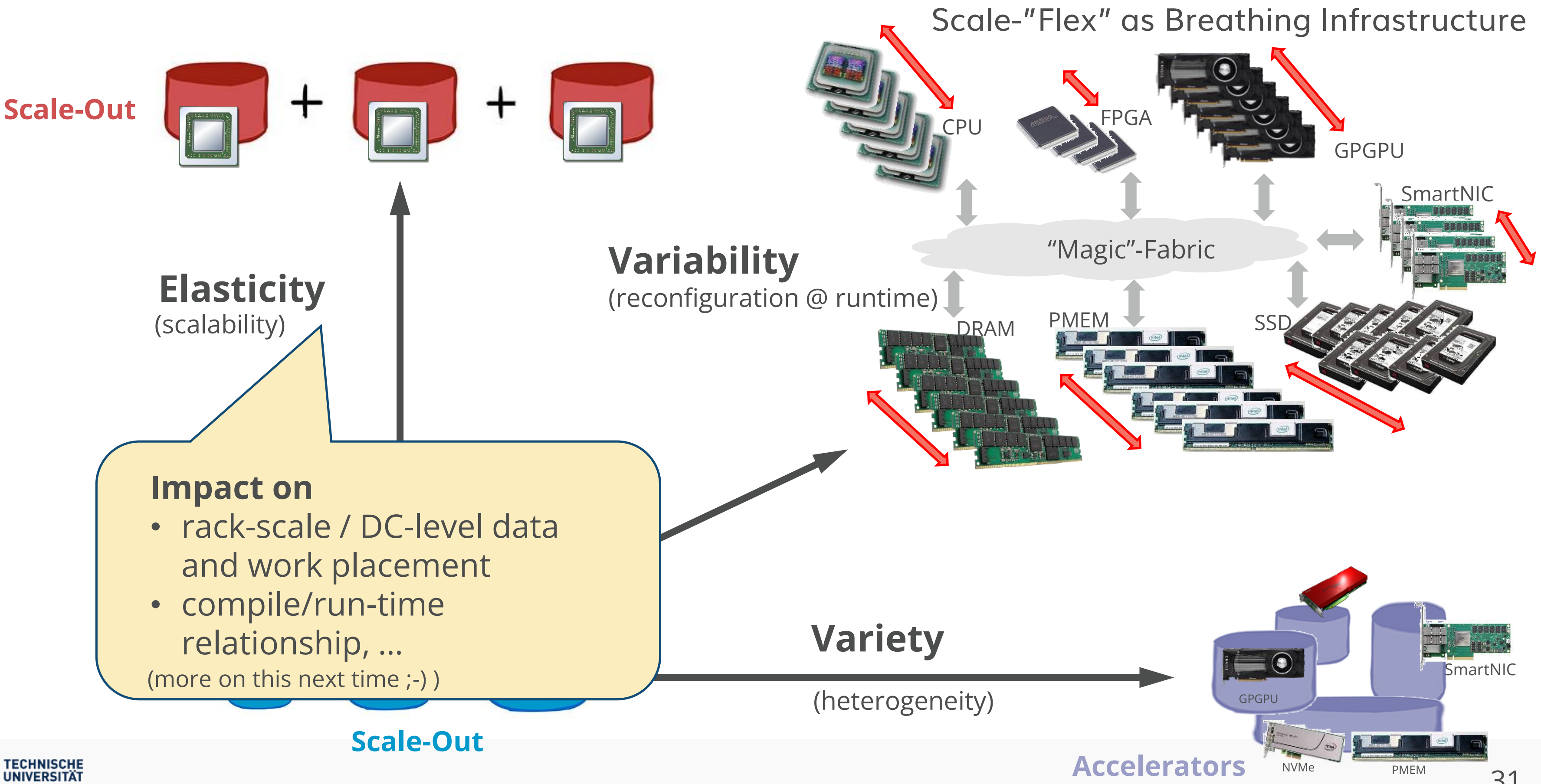
Variety

(heterogeneity)

Scale-Out

Accelerators

Design Space for Data Systems Architectures

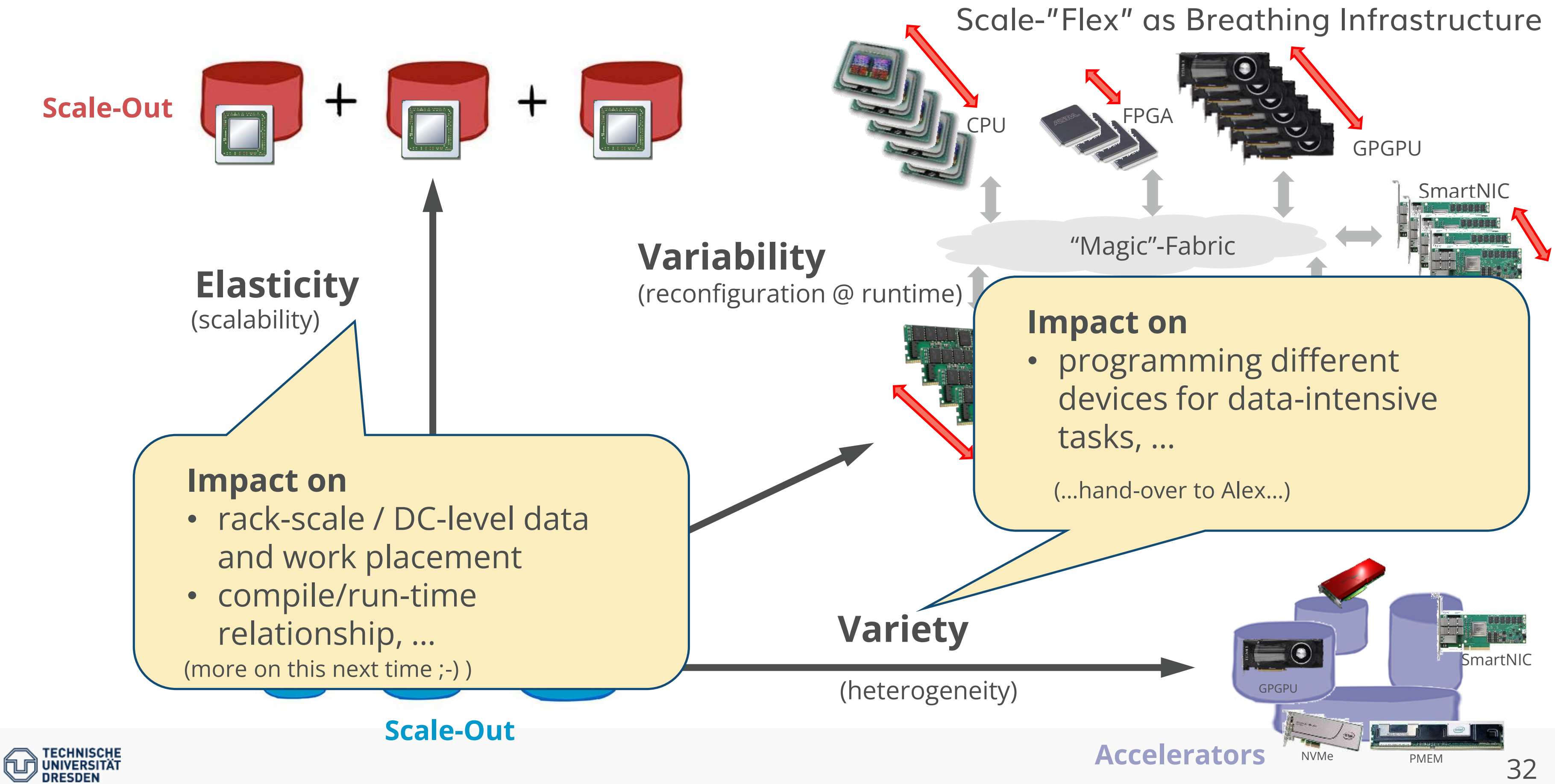


Impact on

- rack-scale / DC-level data and work placement
- compile/run-time relationship, …

(more on this next time ;-))

Design Space for Data Systems Architectures



Impact on

- rack-scale / DC-level data and work placement
- compile/run-time relationship, …

(more on this next time ;-))

Impact on

- programming different devices for data-intensive tasks, …

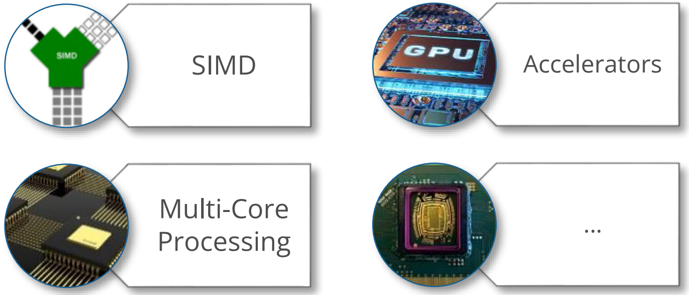
(…hand-over to Alex…)

How to tame Variability?

DB-Systems and Evolving Hardware

Hardware Oblivious Programming

- Plethora of Processing Units
- Heterogeneous Accelerators
- One common paradigm: SIMD
  - In CPUs
  - In GPUs (SIMT, but still!)
  - In FPGAs
  - In our Code…



SIMD

Accelerators

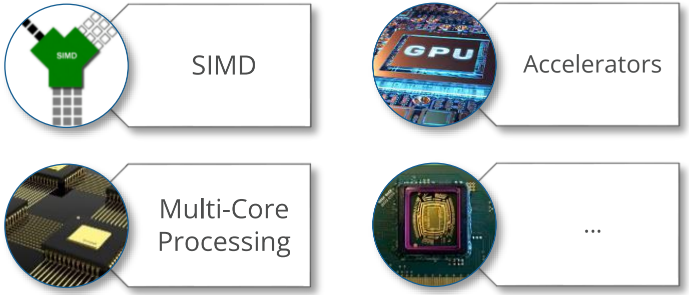
Multi-Core Processing

…

DB-Systems and Evolving Hardware

Hardware Oblivious Programming

- Plethora of Processing Units
- Heterogeneous Accelerators
- One common paradigm: SIMD
  - In CPUs
  - In GPUs (SIMT, but still!)
  - In FPGAs
  - In our Code…


SIMD

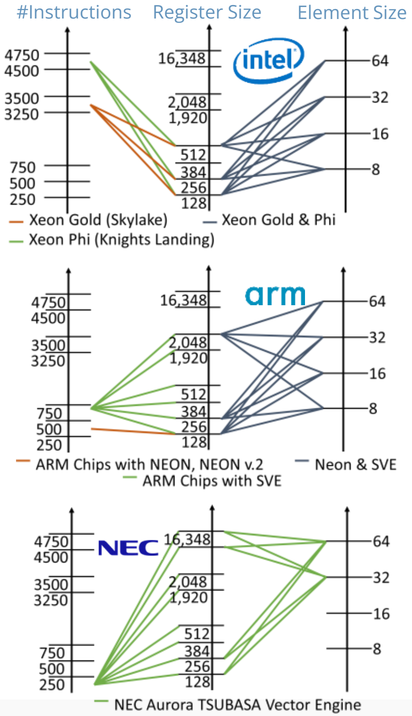
Accelerators

Multi-Core Processing

…

SIMD is heterogeneous in itself

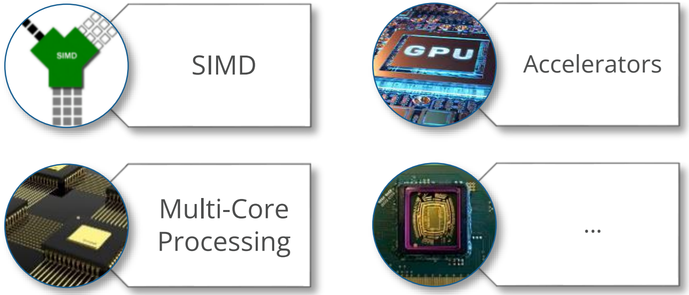
- Varying instructions
- Varying Register types
- Varying effective bitwidths



DB-Systems and Evolving Hardware

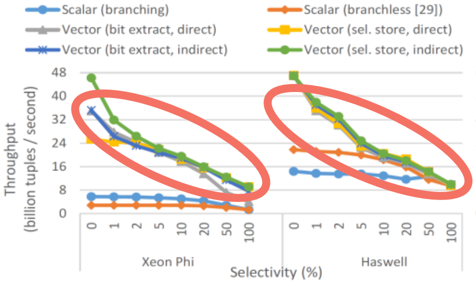
Hardware Oblivious Programming

- Plethora of Processing Units
- Heterogeneous Accelerators
- One common paradigm: SIMD
  - In CPUs
  - In GPUs (SIMT, but still!)
  - In FPGAs
  - In our Code…

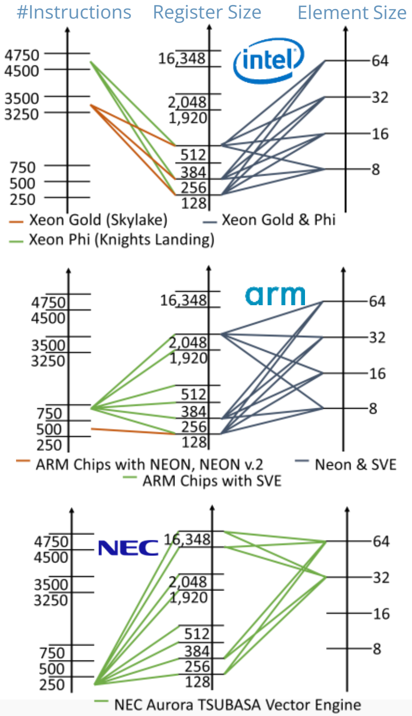


SIMD is heterogeneous in itself

- Varying instructions
- Varying Register types
- Varying effective bitwidths



Variants of selection scan



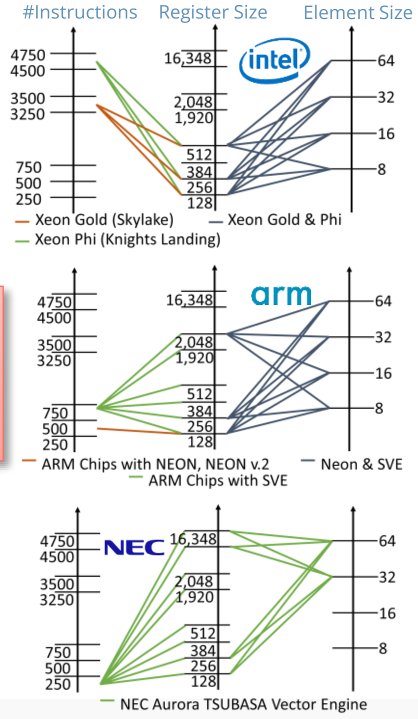
DB-Systems and Evolving Hardware

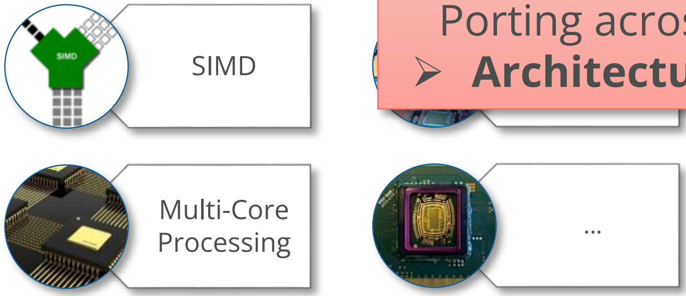
Hardware Oblivious Programming

- Plethora of Processing Units
- Heterogeneous Accelerators
- One common paradigm: SIMD
  - In CPUs
  - In GPUs (SIMT, but still!)
  - In FPGAs
  - In our Code…

SIMD is heterogeneous in itself

- Varying instructions
- Varying Register types
- Varying effective bitwidths

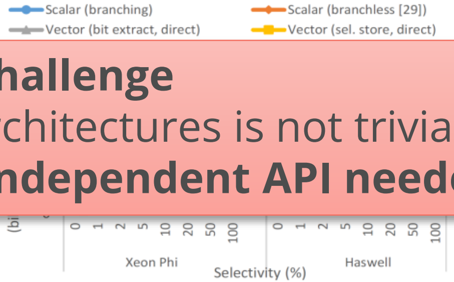




Challenge

Porting across architectures is not trivial!

➢ **Architecture independent API needed**

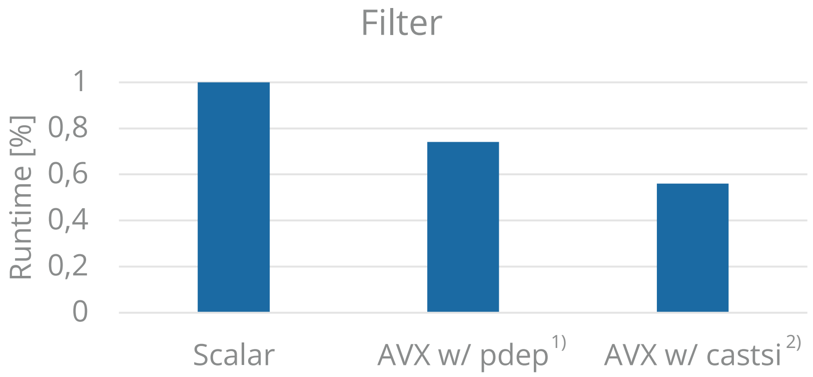


Variants of selection scan

DB-Systems and Evolving Hardware (contd.)

SIMD is not SIMD

- Different intrinsics, same outcome
- Some intrinsics are more costly than others
- Performance is CPU(-architecture) dependent



[1] `_pdep_u64(_mm256_movemask_epi8(bits), mask)`
[2] `_mm256_movemask_pd(_mm256_castsi256_pd(bits))`

DB-Systems and Evolving Hardware (contd.)

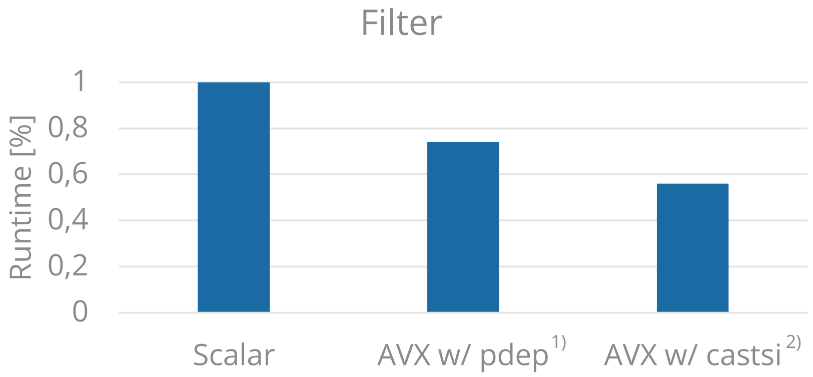
Dresden Database Research Group

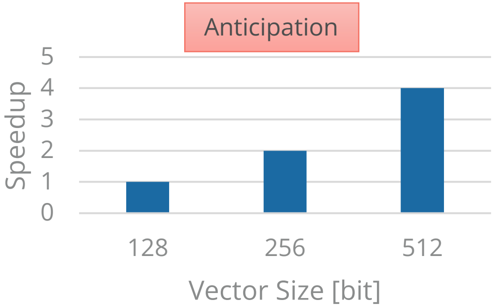
SIMD is not SIMD

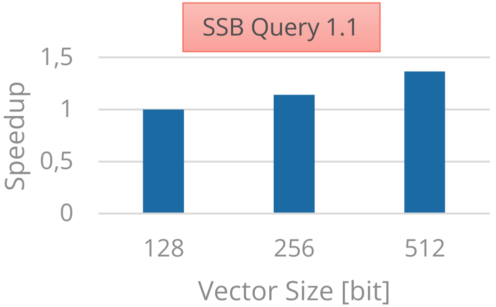
- Different intrinsics, same outcome
- Some intrinsics are more costly than others
- Performance is CPU(-architecture) dependent

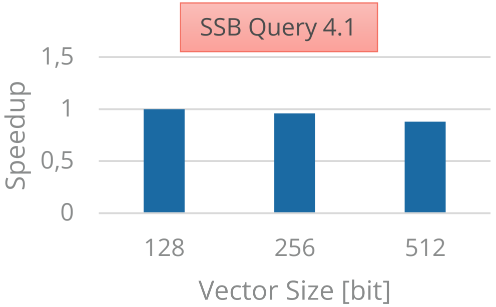
Reality is often disappointing

- Scalability is not linear
- Sometimes wider is worse









[1] `_pdep_u64(_mm256_movemask_epi8(bits), mask)`
[2] `_mm256_movemask_pd(_mm256_castsi256_pd(bits))`

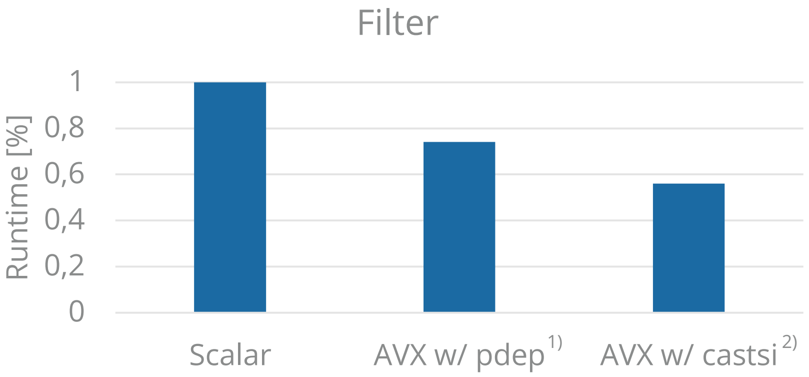
DB-Systems and Evolving Hardware (contd.)

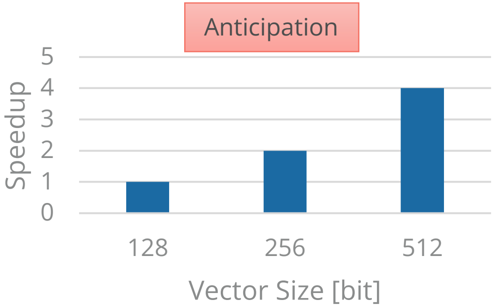
SIMD is not SIMD

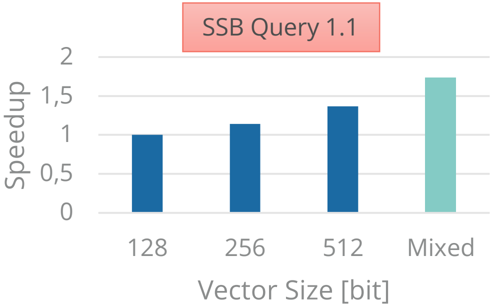
- Different intrinsics, same outcome
- Some intrinsics are more costly than others
- Performance is CPU(-architecture) dependent

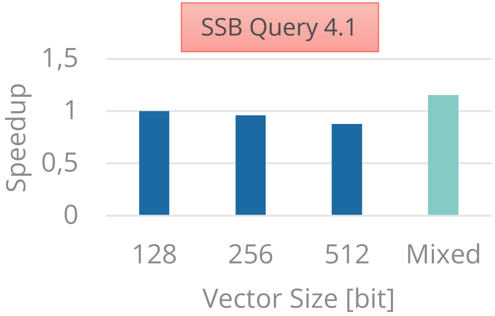
Reality is often disappointing

- Scalability is not linear
- Sometimes wider is worse
- In-Code variety can help!









[1] `_pdep_u64(_mm256_movemask_epi8(bits), mask)`

[2] `_mm256_movemask_pd(_mm256_castsi256_pd(bits))`

DB-Systems and Evolving Hardware (contd.)

Dresden Database Research Group

SIMD is not SIMD

- Different intrinsics, same outcome
- Some intrinsics are more costly than others
- Performance is CPU(-architecture) dependent

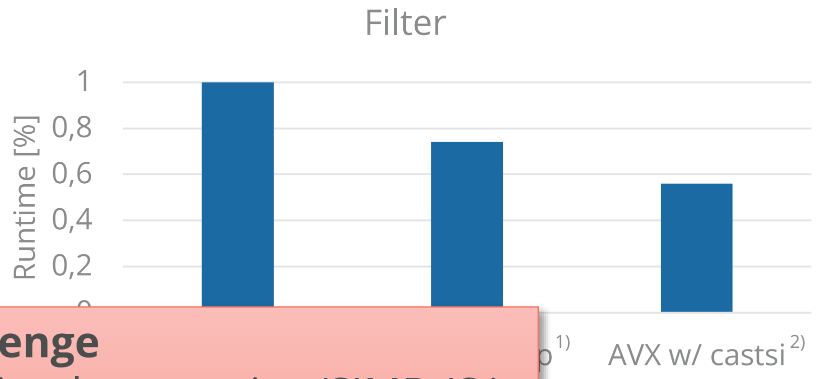
Reality is often disappointing

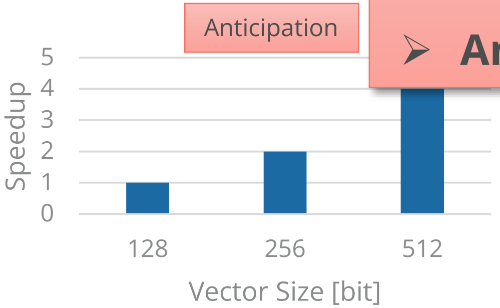
- Scalability is not linear
- Sometimes wider i
- In-Code variety ca

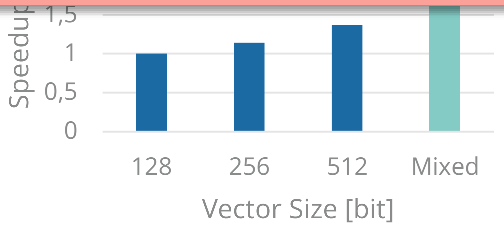
Challenge

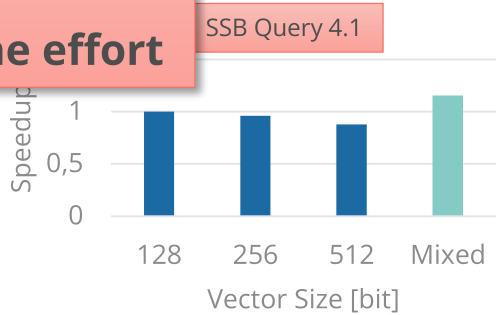
Selecting the appropriate implementation/SIMD ISA is not trivial!

➢ **An abstraction library can alleviate the effort**





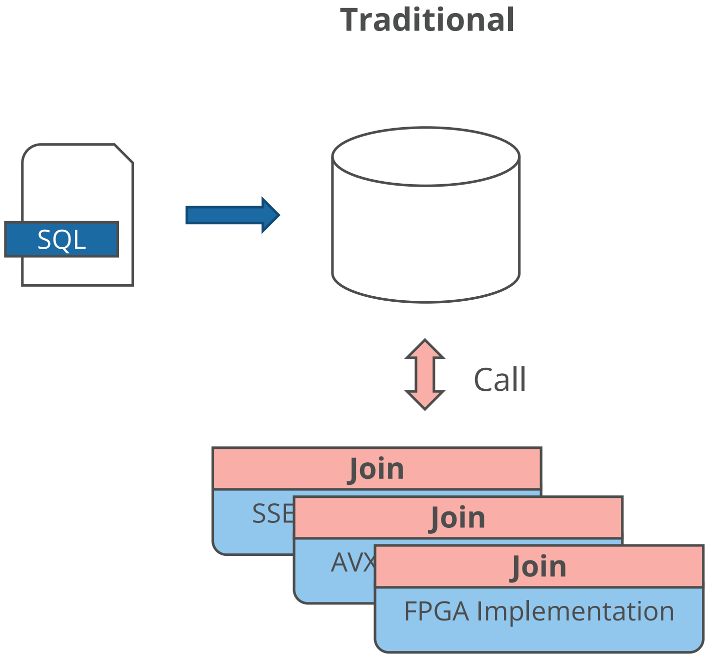




[1] `_pdep_u64(_mm256_movemask_epi8(bits), mask)`

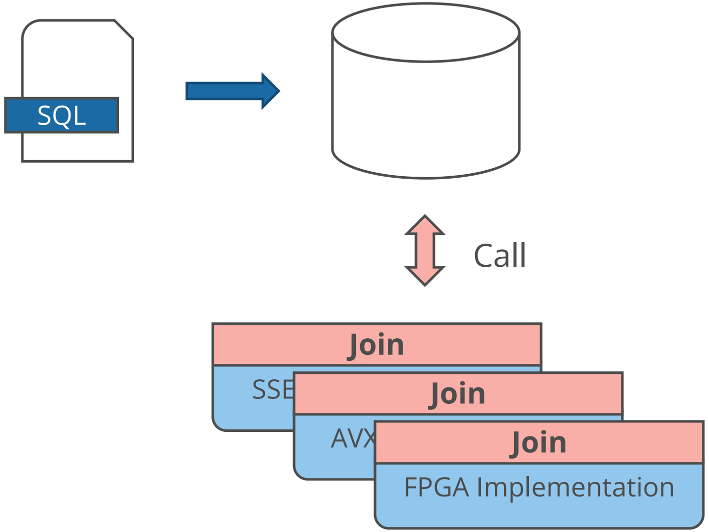
[2] `_mm256_movemask_pd(_mm256_castsi256_pd(bits))`

TSL – Abstracting the Heterogeneity

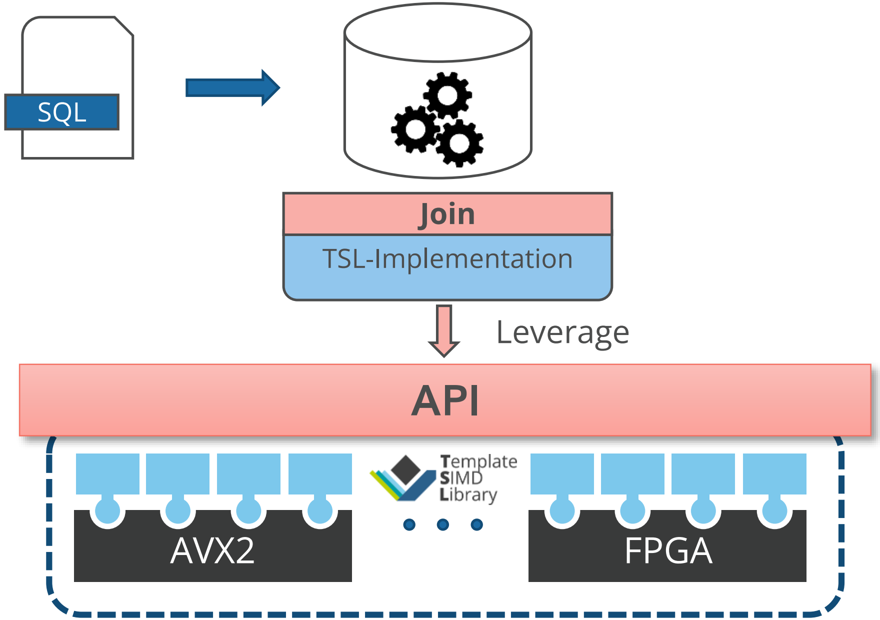


TSL – Abstracting the Heterogeneity

Traditional



Be Flexible!

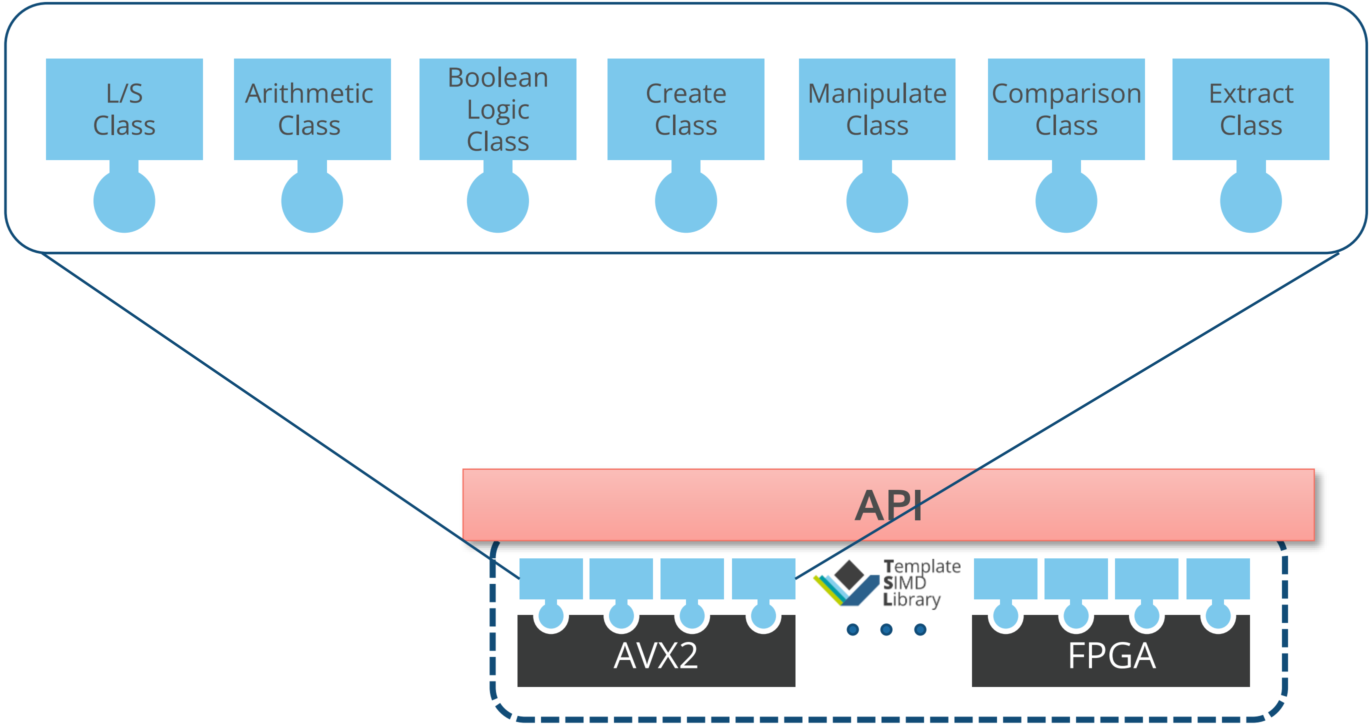


TSL – Abstracting the Heterogeneity

Dresden Database Research Group

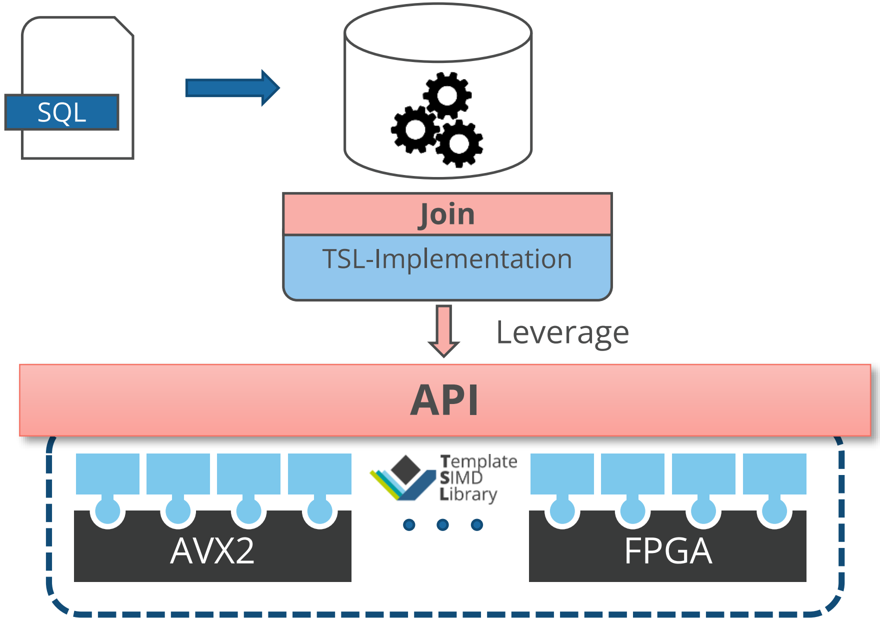
Abstraction Classes

- Data Movement
- Bit Manipulation
- Mask operations
- …

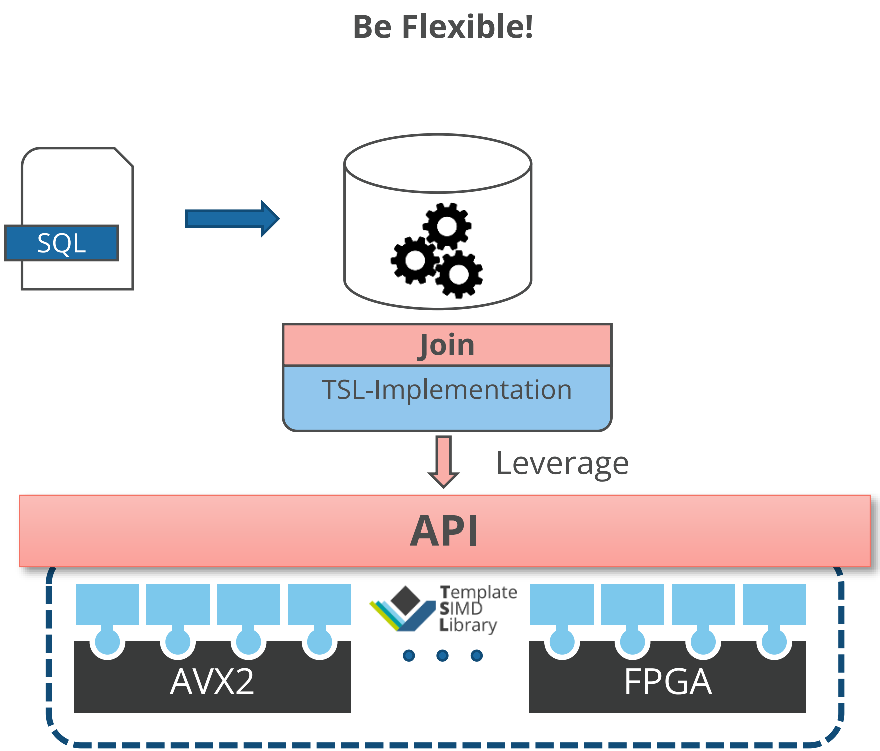


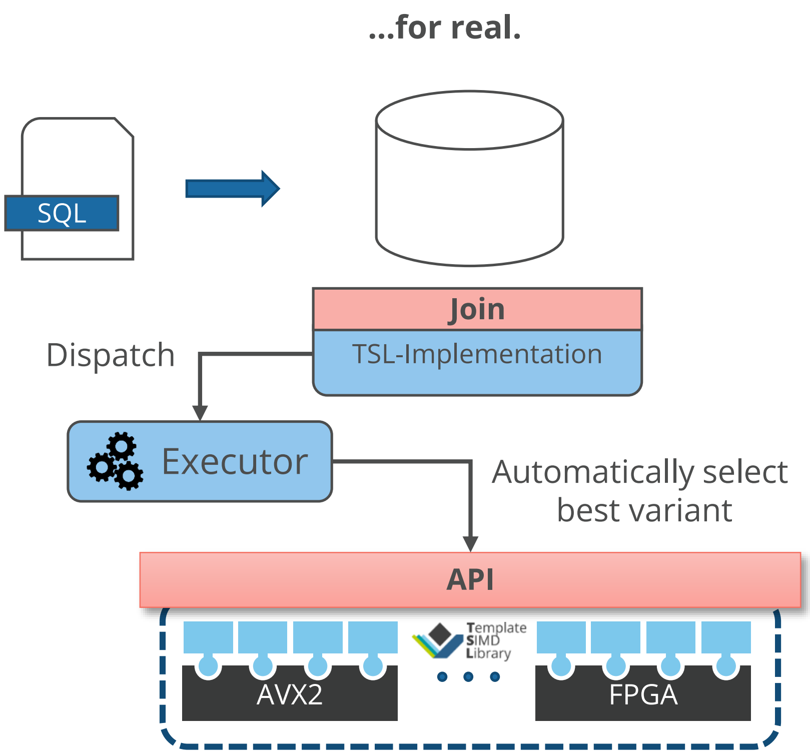
TSL – Abstracting the Heterogeneity

Be Flexible!



TSL – Abstracting the Heterogeneity

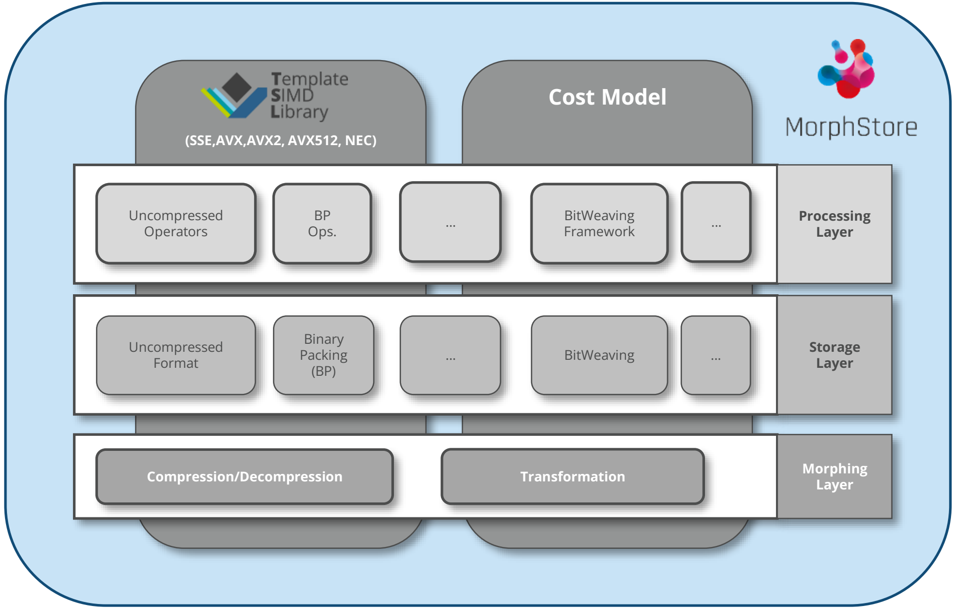




TSL in action: MorphStore

In-Memory Smart Storage System

- **Unique Features**
  - Support for Data Analytics & ML primitives
  - **Compression** and **Vectorization** are first-class citizens
- Support of
  - large corpus of lightweight integer compression algorithms
  - vectorization as programming paradigm
- Compression-aware Query Processing Concept
  - (Micro-) operator-at-a-time Processing Model



P. Damme, A. Ungethüm, J. Pietrzyk, A. Krause, D. Habich, W. Lehner: MorphStore – Analytical Query Engine with a Holistic Compression-Enabled Processing Model. PVLDB 13(11), 2020

Pushing the envelope of SIMD processing: Interpreting FPGAs as SIMD Processing Unit

FPGAs are on the rise

Key-Aspects

- Algorithm-specific integrated circuits
- High performance, while cost-effective

FPGAs are on the rise

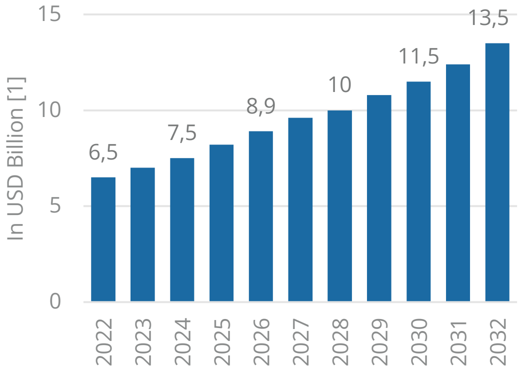
Key-Aspects

- Algorithm-specific integrated circuits
- High performance, while cost-effective

Main differences to traditional ASICs

- Reconfigurable (can adapt to user's application)
- Widely available in the **cloud**

Market Revenue (Forecast)



[1] Data retrieved from https://scoop.market.us/fpga-statistics/; Last accessed: Dec 19, 2023

FPGAs are on the rise

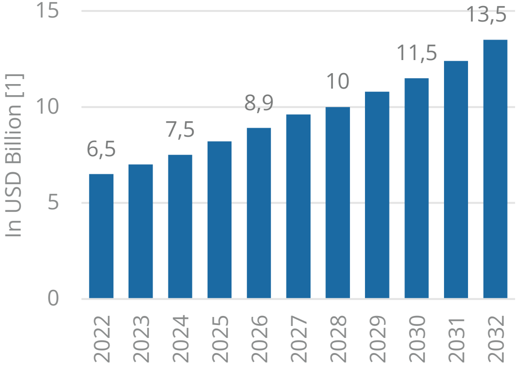
Key-Aspects

- Algorithm-specific integrated circuits
- High performance, while cost-effective

Main differences to traditional ASICs

- Reconfigurable (can adapt to user's application)
- Widely available in the **cloud**

Market Revenue (Forecast)



Programming FPGAs

Gate Level

IP*

[1] Data retrieved from https://scoop.market.us/fpga-statistics/; Last accessed: Dec 19, 2023

* Typically not directly programmed

FPGAs are on the rise

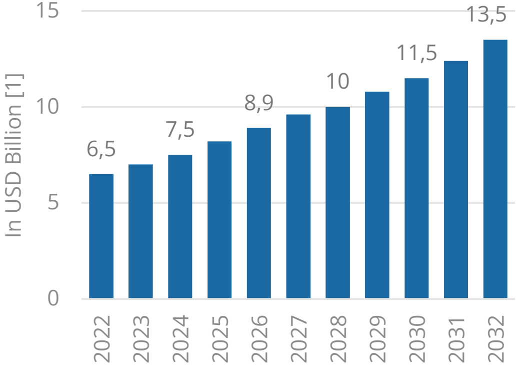
Key-Aspects

- Algorithm-specific integrated circuits
- High performance, while cost-effective

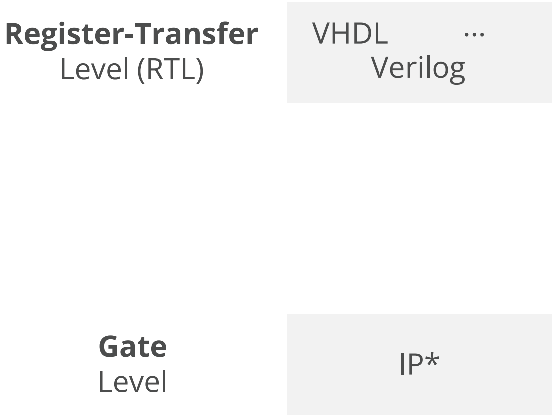
Main differences to traditional ASICs

- Reconfigurable (can adapt to user's application)
- Widely available in the **cloud**

Market Revenue (Forecast)



Programming FPGAs



Register-Transfer Level (RTL)

VHDL ··· Verilog

Gate Level

IP*

[1] Data retrieved from https://scoop.market.us/fpga-statistics/; Last accessed: Dec 19, 2023

* Typically not directly programmed

FPGAs are on the rise

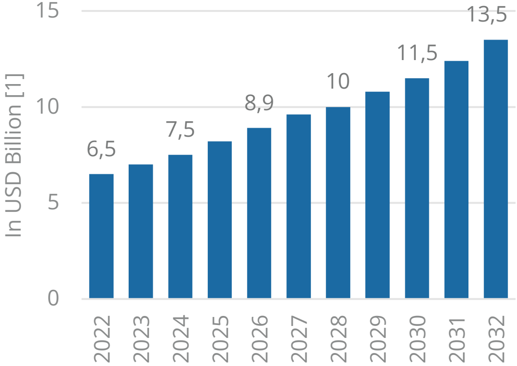
Key-Aspects

- Algorithm-specific integrated circuits
- High performance, while cost-effective

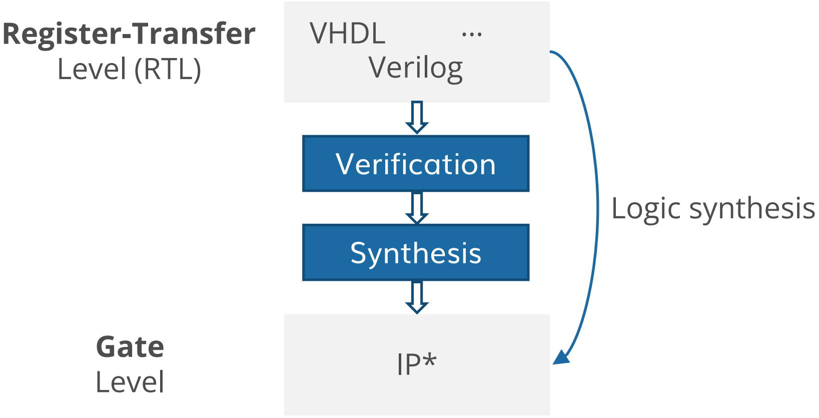
Main differences to traditional ASICs

- Reconfigurable (can adapt to user's application)
- Widely available in the **cloud**

Market Revenue (Forecast)



Programming FPGAs



[1] Data retrieved from https://scoop.market.us/fpga-statistics/; Last accessed: Dec 19, 2023

* Typically not directly programmed

FPGAs are on the rise

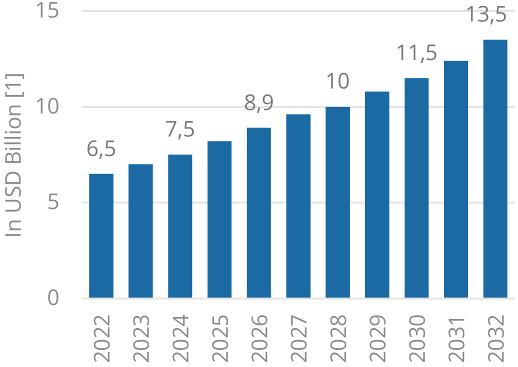
Key-Aspects

- Algorithm-specific integrated circuits
- High performance, while cost-effective

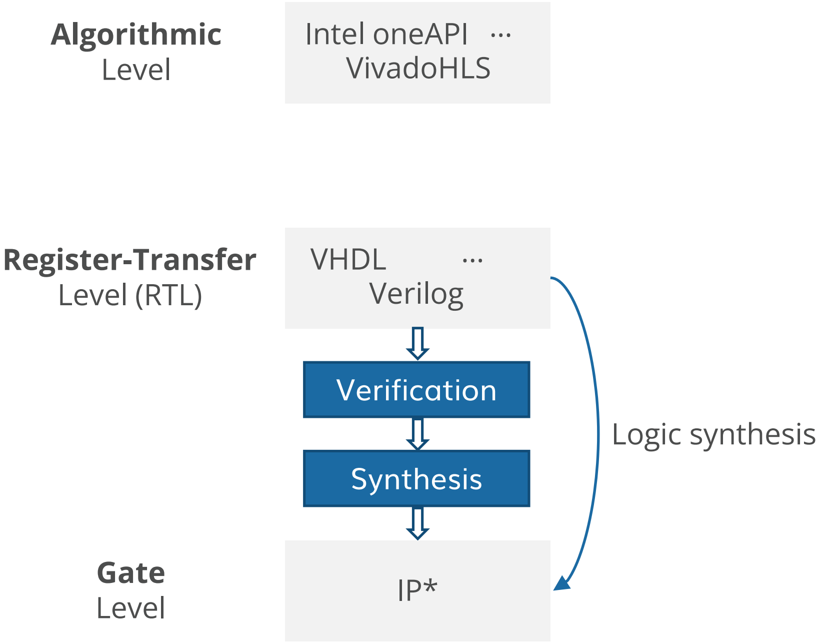
Main differences to traditional ASICs

- Reconfigurable (can adapt to user's application)
- Widely available in the **cloud**

Market Revenue (Forecast)



Programming FPGAs



Algorithmic Level

Intel oneAPI … VivadoHLS

Register-Transfer Level (RTL)

VHDL … Verilog

Verification

Synthesis

Logic synthesis

Gate Level

IP*

[1] Data retrieved from https://scoop.market.us/fpga-statistics/; Last accessed: Dec 19, 2023

* Typically not directly programmed

FPGAs are on the rise

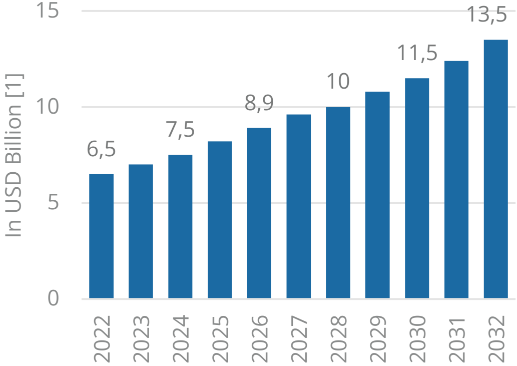
Key-Aspects

- Algorithm-specific integrated circuits
- High performance, while cost-effective

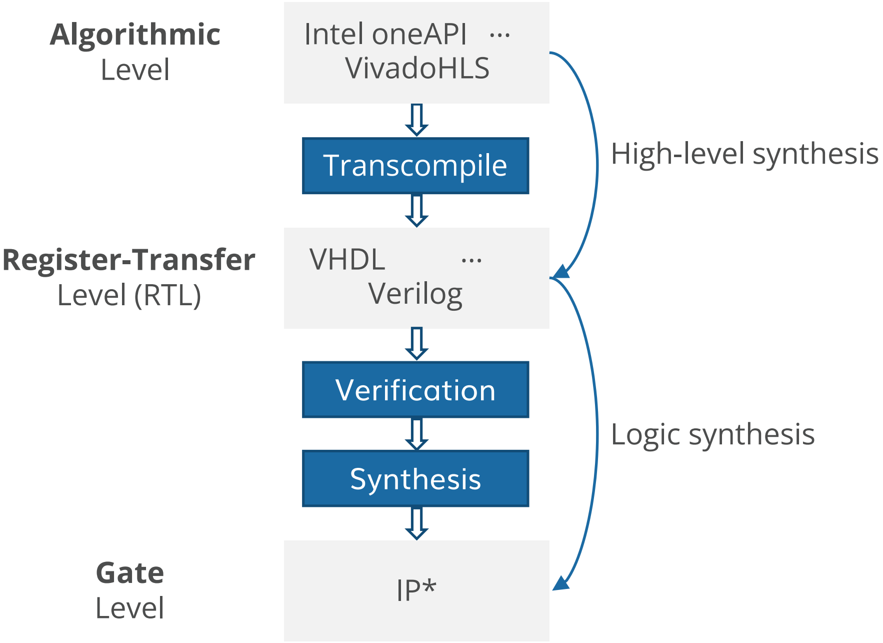
Main differences to traditional ASICs

- Reconfigurable (can adapt to user's application)
- Widely available in the **cloud**

Market Revenue (Forecast)



Programming FPGAs



[1] Data retrieved from https://scoop.market.us/fpga-statistics/; Last accessed: Dec 19, 2023

* Typically not directly programmed

FPGAs are on the rise

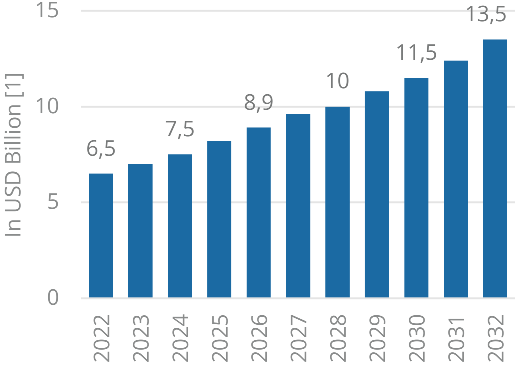
Key-Aspects

- Algorithm-specific integrated circuits
- High performance, while cost-effective

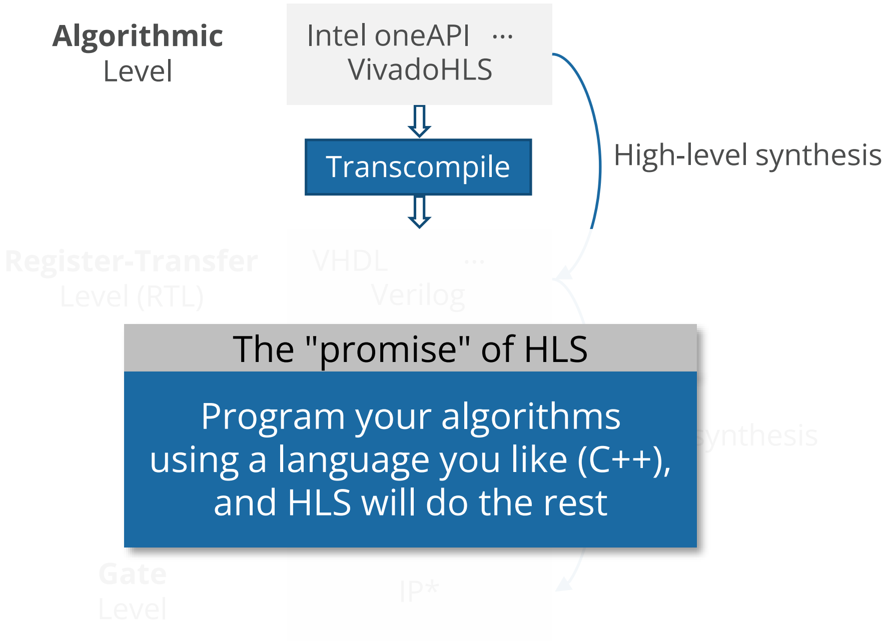
Main differences to traditional ASICs

- Reconfigurable (can adapt to user's application)
- Widely available in the **cloud**

Market Revenue (Forecast)



Programming FPGAs



The "promise" of HLS

Program your algorithms using a language you like (C++), and HLS will do the rest

[1] Data retrieved from https://scoop.market.us/fpga-statistics/; Last accessed: Dec 19, 2023

* Typically not directly programmed

HLS for FPGAs – Case-Study: Intel oneAPI

Dresden Database Research Group

CPU

```
template<typename T, typename PtrT>
void aggregate(PtrT out, PtrT in, size_t n) {
  T result = 0;
  auto const end = in+n;
  for (; in != end; ++in) {
    result += *in;
  }
  *out = result;
}
```

HLS for FPGAs – Case-Study: Intel oneAPI

CPU

```
template<typename T, typename PtrT>
void aggregate(PtrT out, PtrT in, size_t n) {
  T result = 0;
  auto const end = in+n;
  for (; in != end; ++in) {
    result += *in;
  }
  *out = result;
}
```

HLS for FPGAs – Case-Study: Intel oneAPI

CPU

```
template<typename T, typename PtrT>
void aggregate(PtrT out, PtrT in, size_t n) {
  T result = 0;
  auto const end = in+n;
  for (; in != end; ++in) {
    result += *in;
  }
  *out = result;
}
```

HLS for FPGAs – Case-Study: Intel oneAPI

CPU

```
template<typename T, typename PtrT>
void aggregate(PtrT out, PtrT in, size_t n) {
  T result = 0;
  auto const end = in+n;
  for (; in != end; ++in) {
    result += *in;
  }
  *out = result;
}
```

HLS for FPGAs – Case-Study: Intel oneAPI

CPU

```
template<typename T, typename PtrT>
void aggregate(PtrT out, PtrT in, size_t n) {
  T result = 0;
  auto const end = in+n;
  for (; in != end; ++in) {
    result += *in;
  }
  *out = result;
}

int main() {
  using T = uint64_t;
  auto data = (T*)(malloc(128*sizeof(T)));
  auto result = (T*)(malloc(1*sizeof(T)));
  /** init data **/
  aggregate<T>(result, data, 128);
  /** print result, free memory **/
  return 0;
}
```

HLS for FPGAs – Case-Study: Intel oneAPI

CPU

```
template<typename T, typename PtrT>
void aggregate(PtrT out, PtrT in, size_t n) {
  T result = 0;
  auto const end = in+n;
  for (; in != end; ++in) {
    result += *in;
  }
  *out = result;
}

int main() {
  using T = uint64_t;
  auto data = (T*)(malloc(128*sizeof(T)));
  auto result = (T*)(malloc(1*sizeof(T)));
  /** init data **/
  aggregate<T>(result, data, 128);
  /** print result, free memory **/
  return 0;
}
```

FPGA through SYCL/oneAPI

```
void exception_handler(sycl::exception_list e);
int main() {
  using namespace sycl;
  using T = uint64_t;
  auto selector = ext::intel::fpga_selector_v;
  queue q{selector, exception_handler, {}};
  auto data = malloc_host<T>(128*sizeof(T), q);
  auto result = malloc_host<T>(1*sizeof(T), q);
  /** init data **/
  q.submit([&](sycl::handler& h) {
    h.single_task([=]() {
      host_ptr<T> usm_data(data);
      host_ptr<T> usm_result(result);
      aggregate<T>(usm_result, usm_data, 128);
    });
  }).wait();
  /** print result, free memory **/
  return 0;
}
```

HLS for FPGAs – Case-Study: Intel oneAPI

CPU

```
template<typename T, typename PtrT>
void aggregate(PtrT out, PtrT in, size_t n) {
  T result = 0;
  auto const end = in+n;
  for (; in != end; ++in) {
    result += *in;
  }
  *out = result;
}

int main() {
  using T = uint64_t;
  auto data = (T*)(malloc(128*sizeof(T)));
  auto result = (T*)(malloc(1*sizeof(T)));
  /** init data **/
  aggregate<T>(result, data, 128);
  /** print result, free memory **/
  return 0;
}
```

FPGA through SYCL/oneAPI

```
void exception_handler(sycl::exception_list e);
int main() {
  using namespace sycl;
  using T = uint64_t;
  auto selector = ext::intel::fpga_selector_v;
  queue q{selector, exception_handler, {}};
  auto data = malloc_host<T>(128*sizeof(T), q);
  auto result = malloc_host<T>(1*sizeof(T), q);
  /** init data **/
  q.submit([&](sycl::handler& h) {
    h.single_task([=]() {
      host_ptr<T> usm_data(data);
      host_ptr<T> usm_result(result);
      aggregate<T>(usm_result, usm_data, 128);
    });
  }).wait();
  /** print result, free memory **/
  return 0;
}
```

HLS for FPGAs – Case-Study: Intel oneAPI

CPU

```
template<typename T, typename PtrT>
void aggregate(PtrT out, PtrT in, size_t n) {
  T result = 0;
  auto const end = in+n;
  for (; in != end; ++in) {
    result += *in;
  }
  *out = result;
}

int main() {
  using T = uint64_t;
  auto data = (T*)(malloc(128*sizeof(T)));
  auto result = (T*)(malloc(1*sizeof(T)));
  /** init data **/
  aggregate<T>(result, data, 128);
  /** print result, free memory **/
  return 0;
}
```

FPGA through SYCL/oneAPI

```
void exception_handler(sycl::exception_list e);
int main() {
  using namespace sycl;
  using T = uint64_t;
  auto selector = ext::intel::fpga_selector_v;
  queue q{selector, exception_handler, {}};
  auto data = malloc_host<T>(128*sizeof(T), q);
  auto result = malloc_host<T>(1*sizeof(T), q);
  /** init data **/
  q.submit([&](sycl::handler& h) {
    h.single_task([=]() {
      host_ptr<T> usm_data(data);
      host_ptr<T> usm_result(result);
      aggregate<T>(usm_result, usm_data, 128);
    });
  }).wait();
  /** print result, free memory **/
  return 0;
}
```

FPGA Setup, Offloading

Data Setup

HLS for FPGAs – Case-Study: Intel oneAPI

CPU

```
template<typename T, typename PtrT>
void aggregate(PtrT out, PtrT in, size_t n) {
  T result = 0;
  auto const end = in+n;
  for (; in != end; ++in) {
    result += *in;
  }
  *out = result;
}

int main() {
  using T = uint64_t;
  auto data = (T*)(malloc(128*sizeof(T)));
  auto result = (T*)(malloc(1*sizeof(T)));
  /** init data **/
  aggregate<T>(result, data, 128);
  /** print result, free memory **/
  return 0;
}
```

FPGA through SYCL/oneAPI

```
void exception_handler(sycl::exception_list e);
int main() {
  using namespace sycl;
  using T = uint64_t;
  auto selector = ext::intel::fpga_selector_v;
  queue q{selector, exception_handler, {}};
  auto data = malloc_host<T>(128*sizeof(T), q);
  auto result = malloc_host<T>(1*sizeof(T), q);
  /** init data **/
  q.submit([&](sycl::handler& h) {
    h.single_task([=]() {
      host_ptr<T> usm_data(data);
      host_ptr<T> usm_result(result);
      aggregate<T>(usm_result, usm_data, 128);
    });
  }).wait();
  /** print result, free memory **/
  return 0;
}
```

FPGA Setup, Offloading

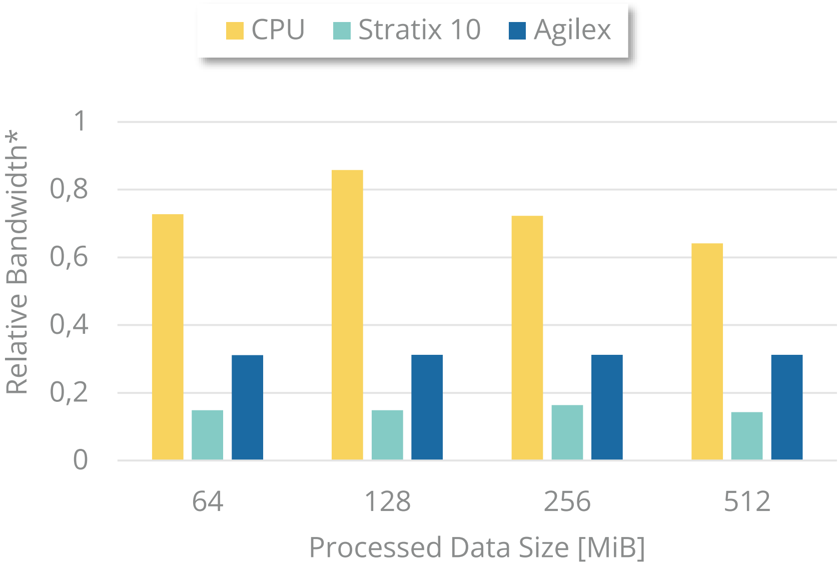
Data Setup

Function Call

HLS for FPGAs – Case-Study: Intel oneAPI

Dresden Database Research Group

Results



FPGA through SYCL/oneAPI

```
void exception_handler(sycl::exception_list e);
int main() {
  using namespace sycl;
  using T = uint64_t;
  auto selector = ext::intel::fpga_selector_v;
  queue q{selector, exception_handler, {}};
  auto data = malloc_host<T>(128*sizeof(T), q);
  auto result = malloc_host<T>(1*sizeof(T), q);
  /** init data **/
  q.submit([&](sycl::handler& h) {
    h.single_task([=]() {
      host_ptr<T> usm_data(data);
      host_ptr<T> usm_result(result);
      aggregate<T>(usm_result, usm_data, 128);
    });
  }).wait();
  /** print result, free memory **/
  return 0;
}
```

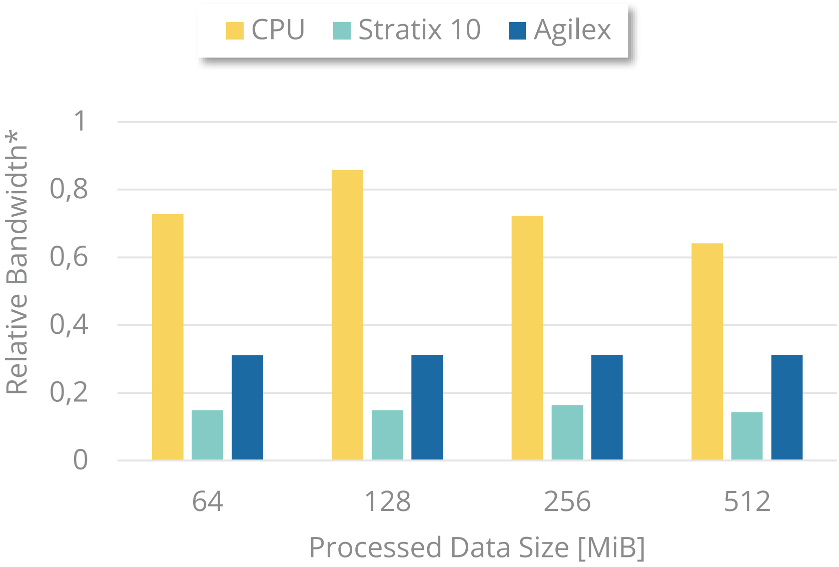
TECHNISCHE UNIVERSITÄT DRESDEN

* max. bandwidths:	CPU	Stratix 10	Agilex
	8.7GB/s (measured)	12.5GB/s (BSP)	17GB/s (BSP)

HLS for FPGAs – Case-Study: Intel oneAPI

Dresden Database Research Group

Results



FPGA through SYCL/oneAPI

```
ion_handler(sycl::exception_list e);
```

Observations

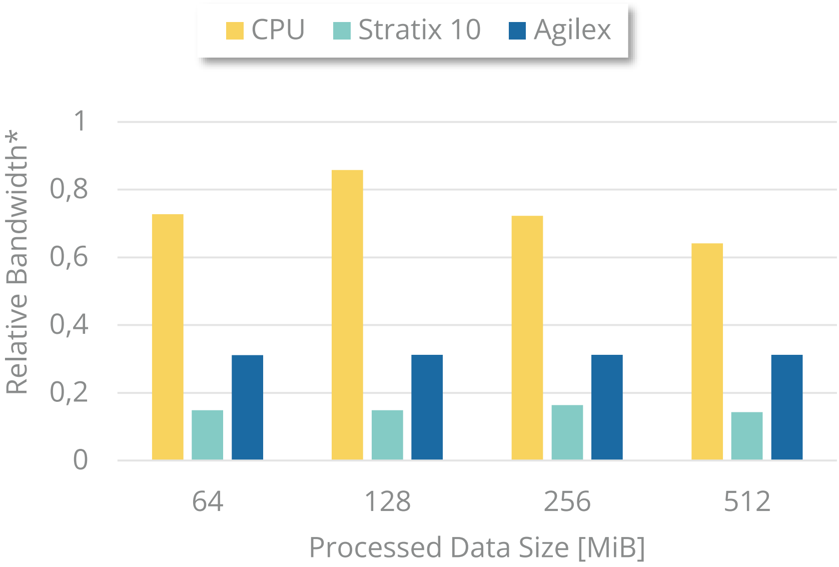
- Manageable effort to compile an **already existing** kernel for FPGA
- No dedicated memory management necessary (FPGA as a co-processor)
- Throughput improvable (underutilization of PCIe)

```
    h.single_task([=]() {
      host_ptr<T> usm_data(data);
      host_ptr<T> usm_result(result);
      aggregate<T>(usm_result, usm_data, 128);
    });
  }).wait();
  /** print result, free memory **/
  return 0;
}
```

* max. bandwidths:	CPU	Stratix 10	Agilex
	8.7GB/s (measured)	12.5GB/s (BSP)	17GB/s (BSP)

HLS for FPGAs – Case-Study: Intel oneAPI

Results



FPGA through SYCL/oneAPI

Observations

- Manageable effort to compile an **already existing** kernel for FPGA
- No dedicated memory management necessary (FPGA as a co-processor)
- Throughput improvable (underutilization of PCIe)

⇩

Improvement Idea

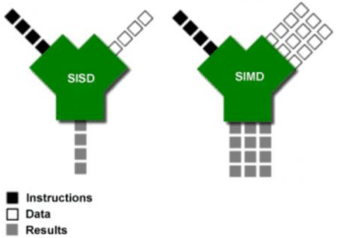
Use PCIe more efficiently,
i.e., all 512 bit instead of 64 bit per cycle
⟶ SIMD is a prime candidate

* max. bandwidths:	CPU	Stratix 10	Agilex
	8.7GB/s (measured)	12.5GB/s (BSP)	17GB/s (BSP)

Intel HLS meets SIMD

What is SIMD?

- Single / same Instruction / operation on Multiple Data



- State-of-the-art technique for improving single thread performance
- Many specialized algorithms and Comprehensive studies for database operators [1]
  - Scans, hashing & probing, …
  - A variety of algorithms, extensively explained

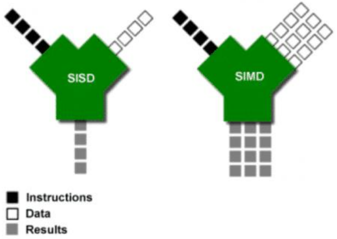
How to use SIMD?

- Using intrinsics
- Using auto-vectorization features

[1] Rethinking SIMD Vectorization for In-Memory Databases. Orestis Polychroniou, Arun Raghavan, Kenneth A. Ross. SIGMOD 2015

Intel HLS meets SIMD

What is SIMD?



- **S**ingle / same
  Instruction / operation on
  Multiple
  Data
- State-of-the-art technique for improving single thread performance
- Many specialized algorithms and Comprehensive studies for database operators [1]
  - Scans, hashing & probing, …
  - A variety of algorithms, extensively explained

How to use SIMD?

- **Using intrinsics**
- Using auto-vectorization features

Aggregation using intrinsics

- Example: AVX512 on unsigned long long

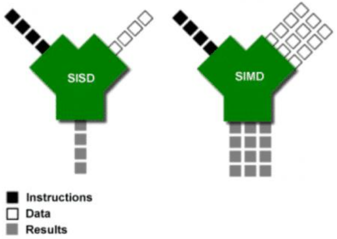
```
void aggregate(
  uint64_t * out, uint64_t const * in, size_t n
) {
  uint64_t result = 0;
  auto const end = in+n;
  for (; in!=end; ++in) {
    result += *in;
  }
  *out = result;
}
```

[1] Rethinking SIMD Vectorization for In-Memory Databases. Orestis Polychroniou, Arun Raghavan, Kenneth A. Ross. SIGMOD 2015

Intel HLS meets SIMD

What is SIMD?

- **S**ingle / same
  Instruction / operation on
  Multiple
  Data



- State-of-the-art technique for improving single thread performance
- Many specialized algorithms and Comprehensive studies for database operators [1]
  - Scans, hashing & probing, …
  - A variety of algorithms, extensively explained

How to use SIMD?

- **Using intrinsics**
- Using auto-vectorization features

Aggregation using intrinsics

- Example: AVX512 on unsigned long long

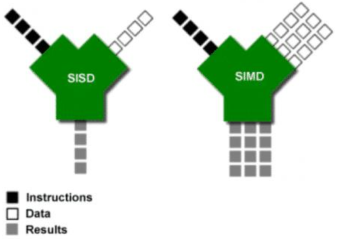
```
void aggregate(
  uint64_t * out, uint64_t const * in, size_t n
) {
  uint64_t result = 0;
  auto const end = in+n;
  for (; in!=end; ++in) {
    result += *in;
  }
  *out = result;
}
```

[1] Rethinking SIMD Vectorization for In-Memory Databases. Orestis Polychroniou, Arun Raghavan, Kenneth A. Ross. SIGMOD 2015

Intel HLS meets SIMD

What is SIMD?

- Single / same
  Instruction / operation on
  Multiple
  Data



- State-of-the-art technique for improving single thread performance
- Many specialized algorithms and Comprehensive studies for database operators [1]
  - Scans, hashing & probing, …
  - A variety of algorithms, extensively explained

How to use SIMD?

- **Using intrinsics**
- Using auto-vectorization features

Aggregation using intrinsics

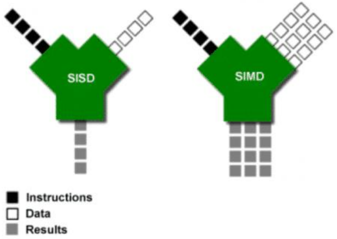
- Example: AVX512 on unsigned long long

```
void aggregate(
  uint64_t * out, uint64_t const * in, size_t n
) {
  __m512i result = _mm512_setzero_si512();
  auto const end = in+n;
  for (; in!=end; ++in) {
    result += *in;
  }
  *out = result;
}
```

[1] Rethinking SIMD Vectorization for In-Memory Databases. Orestis Polychroniou, Arun Raghavan, Kenneth A. Ross. SIGMOD 2015

Intel HLS meets SIMD

What is SIMD?



- **S**ingle / same
  Instruction / operation on
  Multiple
  Data
- State-of-the-art technique for improving single thread performance
- Many specialized algorithms and Comprehensive studies for database operators [1]
  - Scans, hashing & probing, …
  - A variety of algorithms, extensively explained

How to use SIMD?

- **Using intrinsics**
- Using auto-vectorization features

Aggregation using intrinsics

- Example: AVX512 on unsigned long long

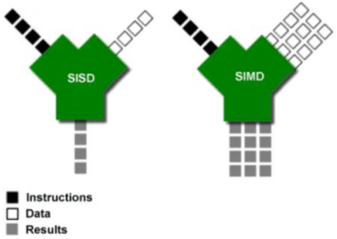
```
void aggregate(
  uint64_t * out, uint64_t const * in, size_t n
) {
  __m512i result = _mm512_setzero_si512();
  auto const end = in+n;
  for (; in!=end; ++in) {
    result += *in;
  }
  *out = result;
}
```

[1] Rethinking SIMD Vectorization for In-Memory Databases. Orestis Polychroniou, Arun Raghavan, Kenneth A. Ross. SIGMOD 2015

Intel HLS meets SIMD

What is SIMD?

- Single / same
  Instruction / operation on
  Multiple
  Data



- State-of-the-art technique for improving single thread performance
- Many specialized algorithms and Comprehensive studies for database operators [1]
  - Scans, hashing & probing, …
  - A variety of algorithms, extensively explained

How to use SIMD?

- **Using intrinsics**
- Using auto-vectorization features

Aggregation using intrinsics

- Example: AVX512 on unsigned long long

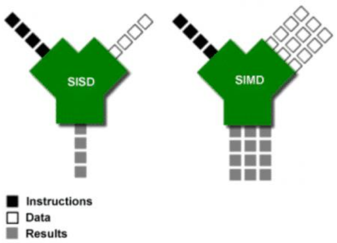
```
void aggregate(
  uint64_t * out, uint64_t const * in, size_t n
) {
  __m512i result = _mm512_setzero_si512();
  auto const end = in+n;
  for (; in!=end; in+=8) {
    __m512i data = _mm512_loadu_si512(in);
    result = _mm512_add_epi64(result, data);
  }
  *out = result;
}
```

[1] Rethinking SIMD Vectorization for In-Memory Databases. Orestis Polychroniou, Arun Raghavan, Kenneth A. Ross. SIGMOD 2015

Intel HLS meets SIMD

What is SIMD?

- Single / same
  Instruction / operation on
  Multiple
  Data



- State-of-the-art technique for improving single thread performance
- Many specialized algorithms and Comprehensive studies for database operators [1]
  - Scans, hashing & probing, …
  - A variety of algorithms, extensively explained

How to use SIMD?

- **Using intrinsics**
- Using auto-vectorization features

Aggregation using intrinsics

- Example: AVX512 on unsigned long long

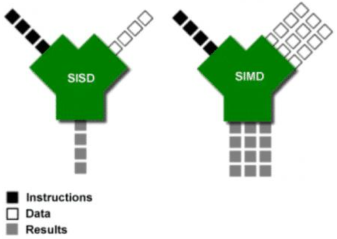
```
void aggregate(
  uint64_t * out, uint64_t const * in, size_t n
) {
  __m512i result = _mm512_setzero_si512();
  auto const end = in+n;
  for (; in!=end; in+=8) {
    __m512i data = _mm512_loadu_si512(in);
    result = _mm512_add_epi64(result, data);
  }
  *out = result;
}
```

[1] Rethinking SIMD Vectorization for In-Memory Databases. Orestis Polychroniou, Arun Raghavan, Kenneth A. Ross. SIGMOD 2015

Intel HLS meets SIMD

What is SIMD?

- Single / same
  Instruction / operation on
  Multiple
  Data



- State-of-the-art technique for improving single thread performance
- Many specialized algorithms and Comprehensive studies for database operators [1]
  - Scans, hashing & probing, …
  - A variety of algorithms, extensively explained

How to use SIMD?

- **Using intrinsics**
- Using auto-vectorization features

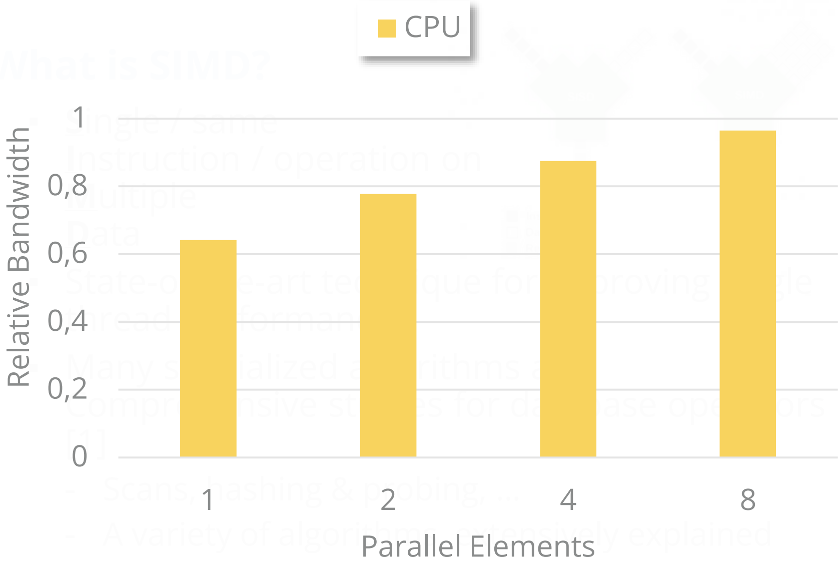
Aggregation using intrinsics

- Example: AVX512 on unsigned long long

```
void aggregate(
  uint64_t * out, uint64_t const * in, size_t n
) {
  __m512i result = _mm512_setzero_si512();
  auto const end = in+n;
  for (; in!=end; in+=8) {
    __m512i data = _mm512_loadu_si512(in);
    result = _mm512_add_epi64(result, data);
  }
  *out = _mm512_reduce_add_epi64(result);
}
```

[1] Rethinking SIMD Vectorization for In-Memory Databases. Orestis Polychroniou, Arun Raghavan, Kenneth A. Ross. SIGMOD 2015

Intel HLS meets SIMD



How to use SIMD?

- **Using intrinsics**
- Using auto-vectorization features

Aggregation using intrinsics

- Example: AVX512 on unsigned long long

```
void aggregate(
  uint64_t * out, uint64_t const * in, size_t n
) {
  __m512i result = _mm512_setzero_si512();
  auto const end = in+n;
  for (; in!=end; in+=8) {
    __m512i data = _mm512_loadu_si512(in);
    result = _mm512_add_epi64(result, data);
  }
  *out = _mm512_reduce_add_epi64(result);
```

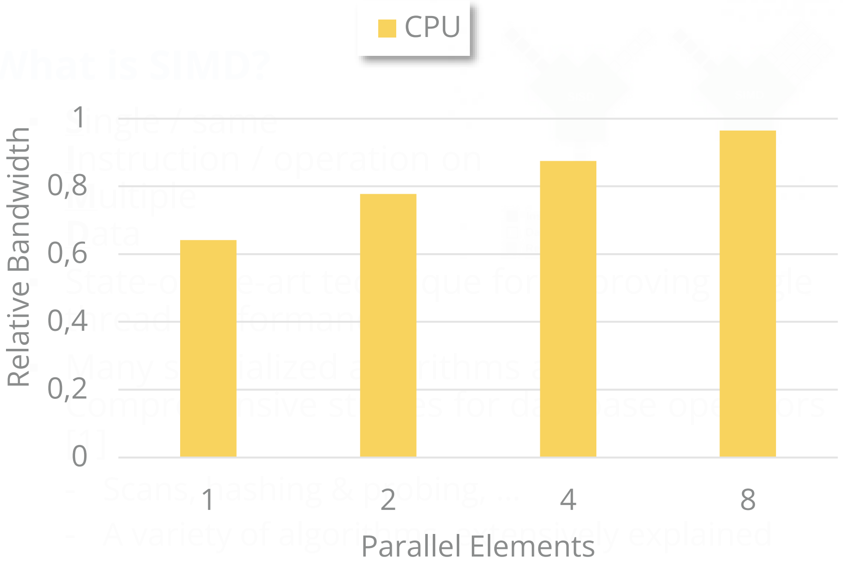
Observations

- Notable improvement of bandwidth utilization
- Higher degree of data-parallelism leads to better throughput

BUT: No intrinsics on FPGA

* max. bandwidths: CPU 8.7GB/s (measured)

Intel HLS meets SIMD



How to use SIMD?

- Using intrinsics
- **Using auto-vectorization features**

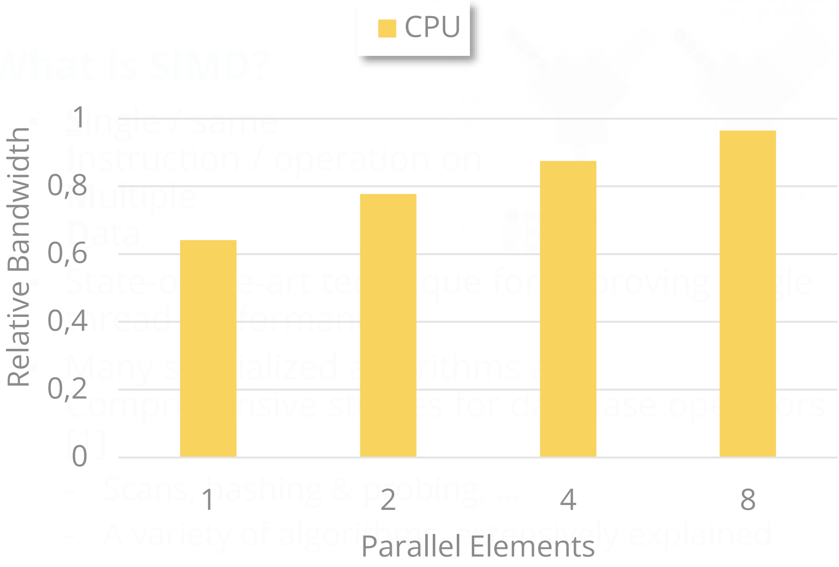
Aggregation using Auto-Vectorization

- Example: 8 unsigned long long parallel

```
void aggregate(
  uint64_t * out, uint64_t const * in, size_t n
) {
  __m512i result = _mm512_setzero_si512();
  auto const end = in+n;
  for (; in!=end; in+=8) {
    __m512i data = _mm512_loadu_si512(in);
    result = _mm512_add_epi64(result, data);
  }
  *out = _mm512_reduce_add_epi64(result);
}
```

* max. bandwidths: CPU 8.7GB/s (measured)

Intel HLS meets SIMD



How to use SIMD?

- Using intrinsics
- **Using auto-vectorization features**

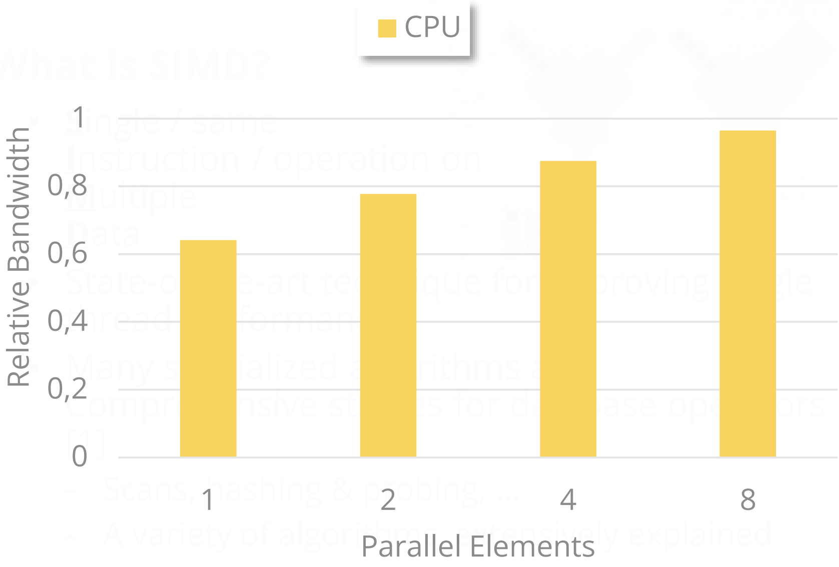
Aggregation using Auto-Vectorization

- Example: 8 unsigned long long parallel

```
void aggregate(
  uint64_t * out, uint64_t const * in, size_t n
) {
  __m512i result = _mm512_setzero_si512();
  auto const end = in+n;
  for (; in!=end; in+=8) {
    __m512i data = _mm512_loadu_si512(in);
    result = _mm512_add_epi64(result, data);
  }
  *out = _mm512_reduce_add_epi64(result);
}
```

* max. bandwidths:	CPU
	8.7GB/s (measured)

Intel HLS meets SIMD



How to use SIMD?

- Using intrinsics
- **Using auto-vectorization features**

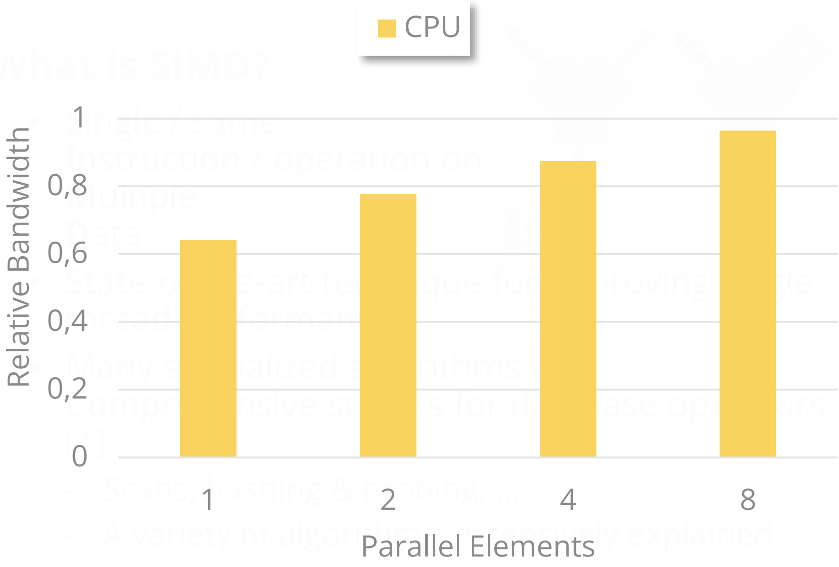
Aggregation using Auto-Vectorization

- Example: 8 unsigned long long parallel

```
void aggregate(
  uint64_t * out, uint64_t const * in, size_t n
) {
  uint64_t results[8] = {};
  auto const end = in+n;
  for (; in!=end; in+=8) {
    __m512i data = _mm512_loadu_si512(in);
    result = _mm512_add_epi64(result, data);
  }
  *out = _mm512_reduce_add_epi64(result);
}
```

* max. bandwidths:	CPU
	8.7GB/s (measured)

Intel HLS meets SIMD



How to use SIMD?

- Using intrinsics
- **Using auto-vectorization features**

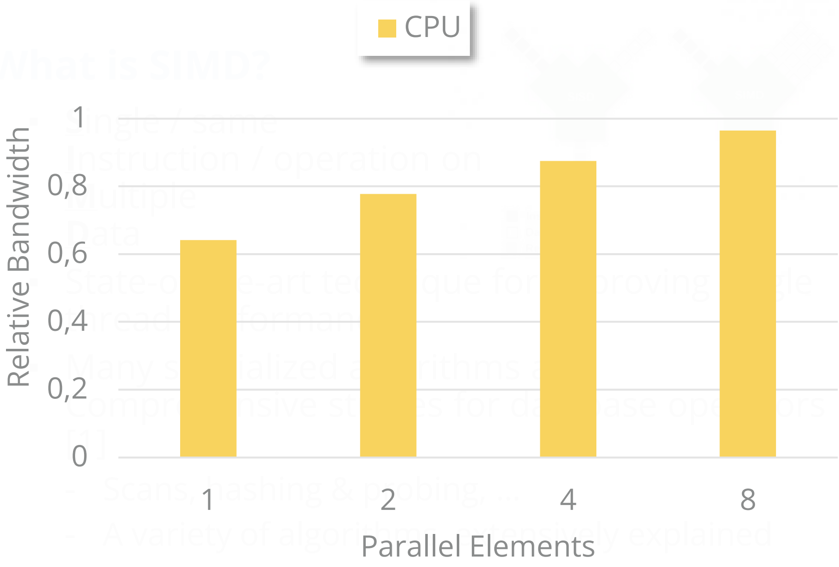
Aggregation using Auto-Vectorization

- Example: 8 unsigned long long parallel

```
void aggregate(
  uint64_t * out, uint64_t const * in, size_t n
) {
  uint64_t results[8] = {};
  auto const end = in+n;
  for (; in!=end; in+=8) {
    __m512i data = _mm512_loadu_si512(in);
    result = _mm512_add_epi64(result, data);
  }
  *out = _mm512_reduce_add_epi64(result);
}
```

* max. bandwidths:	CPU
	8.7GB/s (measured)

Intel HLS meets SIMD



How to use SIMD?

- Using intrinsics
- **Using auto-vectorization features**

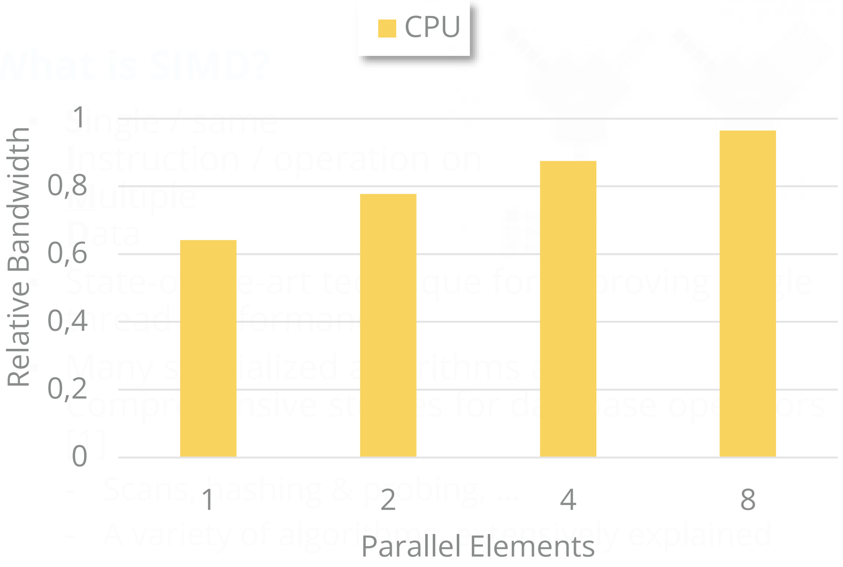
Aggregation using Auto-Vectorization

- Example: 8 unsigned long long parallel

```
void aggregate(
  uint64_t * out, uint64_t const * in, size_t n
) {
  uint64_t results[8] = {}, data[8];
  auto const end = in+n; int i;
  for (; in!=end; in+=8) {
    #pragma unroll
    for (i=0; i<8; ++i) { data[i] = in[i]; }
    result = _mm512_add_epi64(result, data);
  }
  *out = _mm512_reduce_add_epi64(result);
}
```

* max. bandwidths: CPU 8.7GB/s (measured)

Intel HLS meets SIMD



How to use SIMD?

- Using intrinsics
- **Using auto-vectorization features**

Aggregation using Auto-Vectorization

- Example: 8 unsigned long long parallel

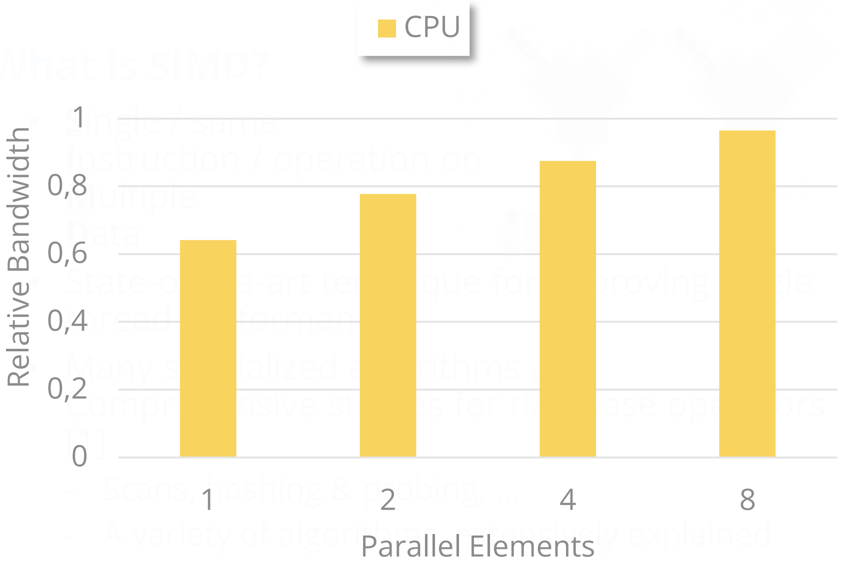
```
void aggregate(
  uint64_t * out, uint64_t const * in, size_t n
) {
  uint64_t results[8] = {}, data[8];
  auto const end = in+n; int i;
  for (; in!=end; in+=8) {
    #pragma unroll
    for (i=0; i<8; ++i) { data[i] = in[i]; }
    result = _mm512_add_epi64(result, data);
  }
  *out = _mm512_reduce_add_epi64(result);
}
```

* max. bandwidths:	CPU
	8.7GB/s (measured)

Intel HLS meets SIMD

What is SIMD?

- Single / same Instruction / operation on Multiple Data
- State-of-the-art technique for improving single thread performance
- Many specialized algorithms and Comprehensive studies for database operators [1]
  - Scans, hashing & probing, …
  - A variety of algorithms extensively explained



How to use SIMD?

- Using intrinsics
- **Using auto-vectorization features**

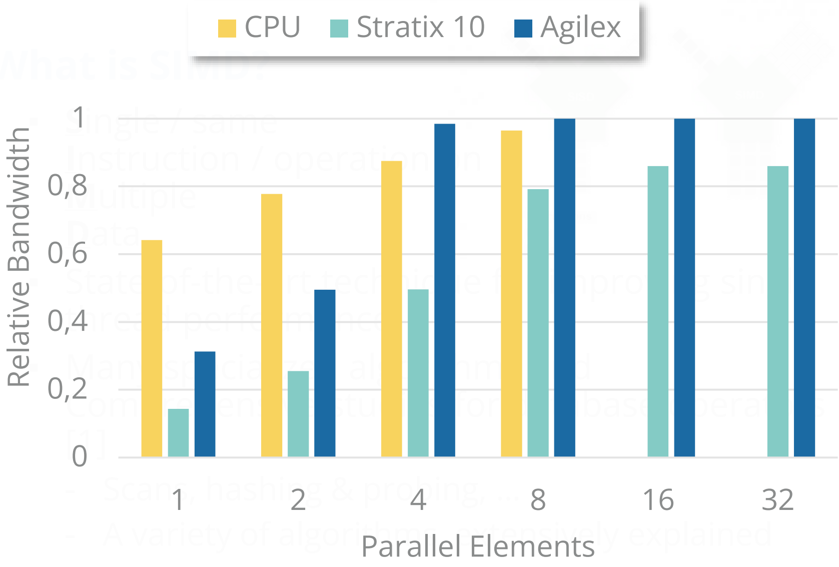
Aggregation using Auto-Vectorization

- Example: 8 unsigned long long parallel

```
void aggregate(
  uint64_t * out, uint64_t const * in, size_t n
) {
  uint64_t results[8] = {}, data[8], result = 0;
  auto const end = in+n; int i;
  for (; in!=end; in+=8) {
    #pragma unroll
    for (i=0; i<8; ++i) { data[i] = in[i]; }
    #pragma unroll
    for (i=0; i<8; ++i) { results[i]+=data[i]; }
  }
  #pragma unroll
  for (i=0; i<8; ++i) { result += results[i]; }
  *out = result;
}
```

* max. bandwidths:	CPU
	8.7GB/s (measured)

Intel HLS meets SIMD



How to use SIMD?

- Using intrinsics
- **Using auto-vectorization features**

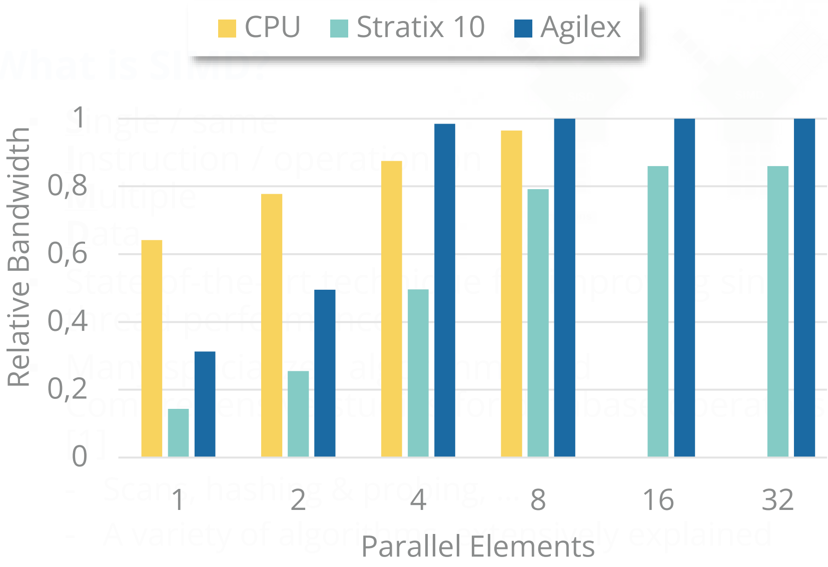
Aggregation using Auto-Vectorization

- Example: 8 unsigned long long parallel

```
void aggregate(
  uint64_t * out, uint64_t const * in, size_t n
) {
  uint64_t results[8] = {}, data[8], result = 0;
  auto const end = in+n; int i;
  for (; in!=end; in+=8) {
    #pragma unroll
    for (i=0; i<8; ++i) { data[i] = in[i]; }
    #pragma unroll
    for (i=0; i<8; ++i) { results[i]+=data[i]; }
  }
  #pragma unroll
  for (i=0; i<8; ++i) { result += results[i]; }
  *out = result;
}
```

* max. bandwidths:	CPU	Stratix 10	Agilex
	8.7GB/s (measured)	12.5GB/s (BSP)	17GB/s (BSP)

Intel HLS meets SIMD



Aggregation using Auto-Vectorization

Observations

- Intrinsics can be substituted with loops, processing fixed-sized arrays
- Loop-unrolling leads to data-parallel execution
- Higher degree of data-level parallelism possible on FGPA compared to CPU (arbitrary size, limited by available ressources)

BUT: Programming autovectorizer-friendly code is laborious and cumbersome

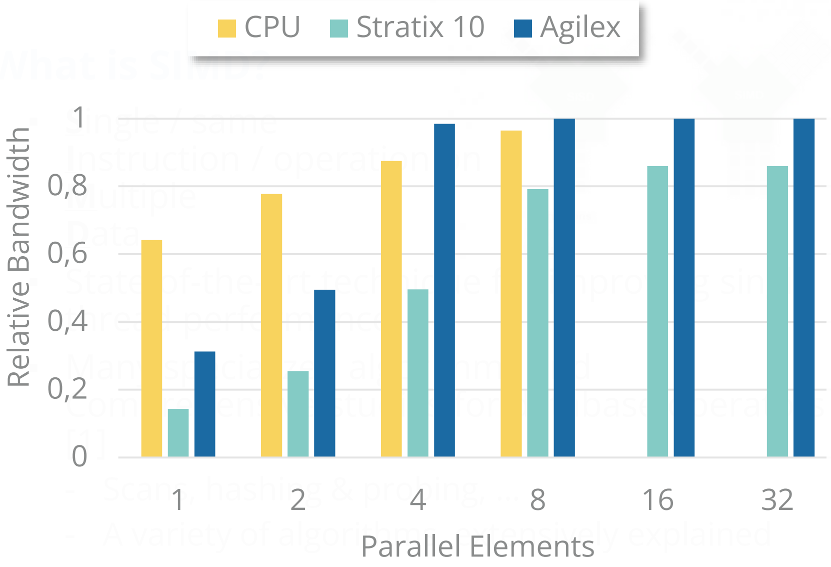
How to use SIMD?

- Using intrinsics
- **Using auto-vectorization features**

* max. bandwidths:	CPU	Stratix 10	Agilex
	8.7GB/s (measured)	12.5GB/s (BSP)	17GB/s (BSP)

Intel HLS meets SIMD

Dresden Database Research Group



How to use SIMD?

- Using intrinsics
- **Using auto-vectorization features**

Aggregation using Auto-Vectorization

Observations

- Intrinsics can be substituted with loops, processing fixed-sized arrays
- Loop-unrolling leads to data-parallel execution
- Higher degree of data-level parallelism possible on FGPA compared to CPU (arbitrary size, limited by available ressources)

BUT: Programming autovectorizer-friendly code is laborious and cumbersome

⇩

Our Idea

Create a custom SIMD Instruction Set for FPGA

* max. bandwidths:	CPU	Stratix 10	Agilex
	8.7GB/s (measured)	12.5GB/s (BSP)	17GB/s (BSP)

Program your (custom) SIMD extension on FPGA

SIMD Types	SIMD Intrinsics	Custom SIMD Intrinsics

Program your (custom) SIMD extension on FPGA

SIMD Types | SIMD Intrinsics | Custom SIMD Intrinsics

```
template<typename T, size_t VSizeBits>
struct intelFPGA {
  constexpr static auto VL() {
    return VSizeBits/(sizeof(T)*CHAR_BIT);
  }
  using reg_t =
    __attribute__((register)) std::array<T, VL()>;
  using mask_t = ac_int<VL(), false>;
  using scalar_int_t = typename
    std::conditional_t<std::is_integral_v<T>,
      ac_int<VSizeBits, false>,
      register_t
  >;
};
template<typename T, size_t VSizeBits>
fpga_reg = typename intelFPGA<T,VSizeBits>::reg_t;
template<typename T, size_t VSizeBits>
fpga_si_reg =
  typename intelFPGA<T,VSizeBits>::scalar_int_t;
template<typename T>
fpga_si = ac_int<sizeof(T)*CHAR_BIT, false>;
```

Program your (custom) SIMD extension on FPGA

SIMD Types | SIMD Intrinsics | Custom SIMD Intrinsics

```
template<typename T, size_t VSizeBits>
struct intelFPGA {
  constexpr static auto VL() {
    return VSizeBits/(sizeof(T)*CHAR_BIT);
  }
  using reg_t =
    __attribute__((register)) std::array<T, VL()>;
  using mask_t = ac_int<VL(), false>;
  using scalar_int_t = typename
    std::conditional_t<std::is_integral_v<T>,
      ac_int<VSizeBits, false>,
      register_t
  >;
};
template<typename T, size_t VSizeBits>
fpga_reg = typename intelFPGA<T,VSizeBits>::reg_t;
template<typename T, size_t VSizeBits>
fpga_si_reg =
  typename intelFPGA<T,VSizeBits>::scalar_int_t;
template<typename T>
fpga_si = ac_int<sizeof(T)*CHAR_BIT, false>;
```

SIMD-Register Definition

- Fundamental building block for SIMD processing
- Strongly typed
- Arbitrary size (= data parallelism)

Program your (custom) SIMD extension on FPGA

SIMD Types | SIMD Intrinsics | Custom SIMD Intrinsics

```
template<typename T, size_t VSizeBits>
struct intelFPGA {
  constexpr static auto VL() {
    return VSizeBits/(sizeof(T)*CHAR_BIT);
  }
  using reg_t =
    __attribute__((register)) std::array<T, VL()>;
  using mask_t = ac_int<VL(), false>;
  using scalar_int_t = typename
    std::conditional_t<std::is_integral_v<T>,
      ac_int<VSizeBits, false>,
      register_t
  >;
};
template<typename T, size_t VSizeBits>
fpga_reg = typename intelFPGA<T,VSizeBits>::reg_t;
template<typename T, size_t VSizeBits>
fpga_si_reg =
  typename intelFPGA<T,VSizeBits>::scalar_int_t;
template<typename T>
fpga_si = ac_int<sizeof(T)*CHAR_BIT, false>;
```

SIMD-Register Definition

- Fundamental building block for SIMD processing
- Strongly typed
- Arbitrary size (= data parallelism)
- Implemented as fixed-sized array

Program your (custom) SIMD extension on FPGA

Dresden Database Research Group

SIMD Types | SIMD Intrinsics | Custom SIMD Intrinsics

```
template<typename T, size_t VSizeBits>
struct intelFPGA {
  constexpr static auto VL() {
    return VSizeBits/(sizeof(T)*CHAR_BIT);
  }
  using reg_t =
    __attribute__((register)) std::array<T, VL()>;
  using mask_t = ac_int<VL(), false>;
  using scalar_int_t = typename
    std::conditional_t<std::is_integral_v<T>,
      ac_int<VSizeBits, false>
      register_t
  >;
};
template<typename T, size_t VSizeBits>
fpga_reg = typename intelFPGA<T,VSizeBits>::reg_t;
template<typename T, size_t VSizeBits>
fpga_si_reg =
  typename intelFPGA<T,VSizeBits>::scalar_int_t;
template<typename T>
fpga_si = ac_int<sizeof(T)*CHAR_BIT, false>;
```

SIMD-Register Definition

- Fundamental building block for SIMD processing
- Strongly typed
- Arbitrary size (= data parallelism)
- Implemented as fixed-sized array

Mask Definition

- Fundamental build block for enabling/disabling specific elements in a SIMD-register
- Using SYCL "Algorithmic C-Type"
  - "Arbitrary-Length" support for wider registers (>512 bit)

Scalar-Integer Register

- Single "large" integral value to support computations across lanes
- Using SYCL AC-Types

Program your (custom) SIMD extension on FPGA

```
void aggregate(
  uint64_t * out, uint64_t const * in, size_t n
) {
  uint64_t results[8] = {}, data[8], result = 0;
  auto const end = in+n; int i;
  for (; in!=end; in+=8) {
    #pragma unroll
    for (i=0; i<8; ++i) { data[i] = in[i]; }
    #pragma unroll
    for (i=0; i<8; ++i) { results[i]+=data[i]; }
  }
  #pragma unroll
  for (i=0; i<8; ++i) { result += results[i]; }
  *out = result;
}
```

Program your (custom) SIMD extension on FPGA

SIMD Types | SIMD Intrinsics | Custom SIMD Intrinsics

```
void aggregate(
  uint64_t * out, uint64_t const * in, size_t n
) {
  uint64_t results[8] = {}, data[8], result = 0;
  auto const end = in+n; int i;
  for (; in!=end; in+=8) {
    #pragma unroll
    for (i=0; i<8; ++i) { data[i] = in[i]; }
    #pragma unroll
    for (i=0; i<8; ++i) { results[i]+=data[i]; }
  }
  #pragma unroll
  for (i=0; i<8; ++i) { result += results[i]; }
  *out = result;
}
```

Memory Access Intrinsics

- Transfer data from memory to register type (and back)

Program your (custom) SIMD extension on FPGA

SIMD Types | SIMD Intrinsics | Custom SIMD Intrinsics

```
void aggregate(
  uint64_t * out, uint64_t const * in, size_t n
) {
  uint64_t results[8] = {}, data[8], result = 0;
  auto const end = in+n; int i;
  for (; in!=end; in+=8) {
    #pragma unroll
    for (i=0; i<8; ++i) { data[i] = in[i]; }
    #pragma unroll
    for (i=0; i<8; ++i) { results[i]+=data[i]; }
  }
  #pragma unroll
  for (i=0; i<8; ++i) { result += results[i]; }
  *out = result;
}
```

Memory Access Intrinsics

- Transfer data from memory to register type (and back)

Element-wise Intrinsics

- Process every element in a register independent from all others

Program your (custom) SIMD extension on FPGA

SIMD Types | SIMD Intrinsics | Custom SIMD Intrinsics

```
void aggregate(
  uint64_t * out, uint64_t const * in, size_t n
) {
  uint64_t results[8] = {}, data[8], result = 0;
  auto const end = in+n; int i;
  for (; in!=end; in+=8) {
    #pragma unroll
    for (i=0; i<8; ++i) { data[i] = in[i]; }
    #pragma unroll
    for (i=0; i<8; ++i) { results[i]+=data[i]; }
  }
  #pragma unroll
  for (i=0; i<8; ++i) { result += results[i]; }
  *out = result;
}
```

Memory Access Intrinsics

- Transfer data from memory to register type (and back)

Element-wise Intrinsics

- Process every element in a register independent from all others

Horizontal Intrinsics

- Reduce all elements of a register into a single scalar value

Program your (custom) SIMD extension on FPGA

SIMD Types | SIMD Intrinsics | Custom SIMD Intrinsics

```
#define INLINE __attribute__((always_inline)) inline
template<typename T, size_t VSizeBits>
INLINE auto load(T const * memory) {
  fpga_reg<T,VSizeBits>::reg_t result{};
  #pragma unroll
  for (auto i : indices(result)) {
    result[i] = memory[i];
  }
  return result;
}
```

General Design Decisions

- Intrinsics are inlined (to be densely packed)

Program your (custom) SIMD extension on FPGA

Dresden Database Research Group

SIMD Types | SIMD Intrinsics | Custom SIMD Intrinsics

```
#define INLINE __attribute__((always_inline)) inline
template<typename T, size_t VSizeBits>
INLINE auto load(T const * memory) {
  fpga_reg<T,VSizeBits>::reg_t result{};
  #pragma unroll
  for (auto i : indices(result)) {
    result[i] = memory[i];
  }
  return result;
}
```

General Design Decisions

- Intrinsics are inlined (to be densely packed)
- Template functions for flexibility at programming time:
  - Arbitrary (arithmetic) type
  - Arbitrary register size

Program your (custom) SIMD extension on FPGA

SIMD Types | SIMD Intrinsics | Custom SIMD Intrinsics

```
#define INLINE __attribute__((always_inline)) inline
template<typename T, size_t VSizeBits>
INLINE auto load(T const * memory) {
  fpga_reg<T,VSizeBits>::reg_t result{};
  #pragma unroll
  for (auto i : indices(result)) {
    result[i] = memory[i];
  }
  return result;
}
```

General Design Decisions

- Intrinsics are inlined (to be densely packed)
- Template functions for flexibility at programming time:
  - Arbitrary (arithmetic) type
  - Arbitrary register size

Memory Access Intrinsics

- Transfer data from memory to register type (and back)
- Following default pattern of loop unrolling
- Zero-copy through copy elision (nrvo)

Program your (custom) SIMD extension on FPGA

```
#define INLINE __attribute__((always_inline)) inline
template<typename T, size_t VSizeBits>
INLINE auto load(T const * memory) {
  fpga_reg<T,VSizeBits>::reg_t result{};
  #pragma unroll
  for (auto i : indices(result)) {
    result[i] = memory[i];
  }
  return result;
}
```

```
template<typename T, size_t VSizeBits>
INLINE auto modulo(
 fpga_reg<T,VSizeBits>::reg_t data, T const modulus
) {
  fpga_reg<T,VSizeBits>::reg_t result{};
  #pragma unroll
  for (auto i : indices(result)) {
    result[i] = data[i] % modulus;
  }
  return result;
}
```

General Design Decisions

- Intrinsics are inlined (to be densely packed)
- Template functions for flexibility at programming time:
  - Arbitrary (arithmetic) type
  - Arbitrary register size

Element-wise Intrinsics

- Process every element in a register independent from all others

Program your (custom) SIMD extension on FPGA

```
#define INLINE __attribute__((always_inline)) inline
template<typename T, size_t VSizeBits>
INLINE auto load(T const * memory) {
  fpga_reg<T,VSizeBits>::reg_t result{};
  #pragma unroll
  for (auto i : indices(result)) {
    result[i] = memory[i];
  }
  return result;
}
template<typename T, size_t VSizeBits>
INLINE auto modulo(
 fpga_reg<T,VSizeBits>::reg_t data, T const modulus
) {
  fpga_reg<T,VSizeBits>::reg_t result{};
  #pragma unroll
  for (auto i : indices(result)) {
    result[i] = data[i] % modulus;
  }
  return result;
}
```

General Design Decisions

- Intrinsics are inlined (to be densely packed)
- Template functions for flexibility at programming time:
  - Arbitrary (arithmetic) type
  - Arbitrary register size

Element-wise Intrinsics

- Process every element in a register independent from all others
- Straight-forward implementation of C++-builtin unary and binary operations (e.g., add, mod,…)[1]

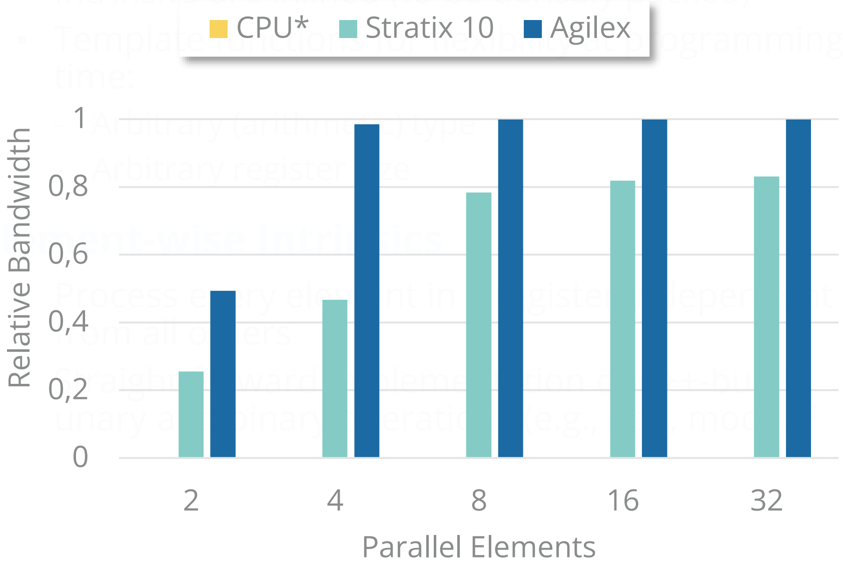
[1] More exotic functionality (e.g., leading-zero-count) presented in the paper

Program your (custom) SIMD extension on FPGA

Dresden Database Research Group

SIMD Types | **SIMD Intrinsics** | Custom SIMD Intrinsics

```
#define INLINE __attribute__((always_inline)) inline
template<typename T, size_t VSizeBits>
INLINE auto load(T const * memory) {
  fpga_reg<T,VSizeBits>::reg_t result{};
  #pragma unroll
  for (auto i : indices(result)) {
    result[i] = memory[i];
  }
  return result;
}
template<typename T, size_t VSizeBits>
INLINE auto modulo(
 fpga_reg<T,VSizeBits>::reg_t data, T const modulus
) {
  fpga_reg<T,VSizeBits>::reg_t result{};
  #pragma unroll
  for (auto i : indices(result)) {
    result[i] = data[i] % modulus;
  }
  return result;
}
```



* no builtin support for Intel CPUs

1 More exotic functionality (e.g., leading-zero-count) presented in the paper

Program your (custom) SIMD extension on FPGA

SIMD Types

SIMD Intrinsics

Custom SIMD Intrinsics

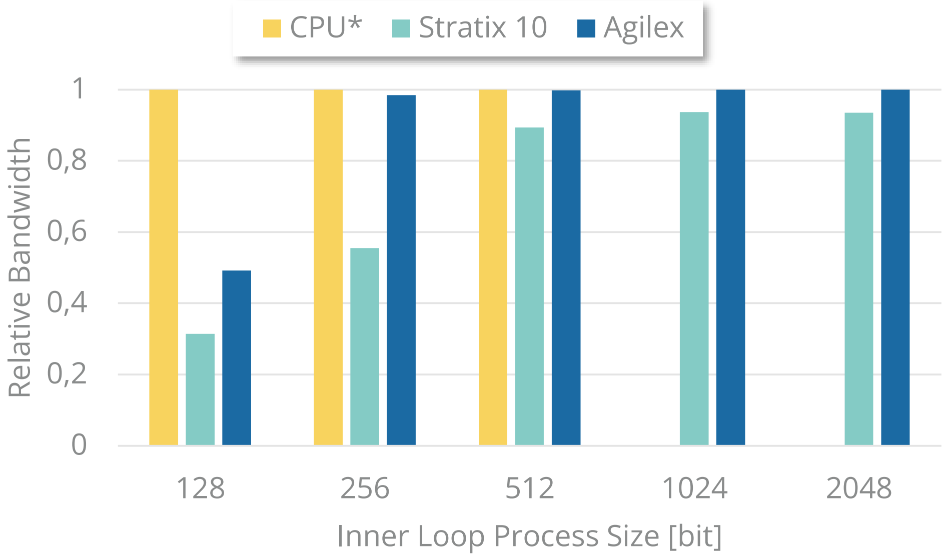
```
template<typename T, size_t VSizeBits>
inline auto clz(fpga_reg<T,VSizeBits> src) {
  fpga_reg<T,VSizeBits> result{};
  auto const bitsize = sizeof(T)*CHAR_BIT;
  using bitseq = ac_int<bitsize,false>;
  #pragma unroll
  for (auto i : indices(result)) {
    bitseq value(src[i]);
    int pos = bitsize - 1;
    #pragma unroll
    for(; pos>=0 && value[pos] == 0; pos--){};
    result[i] = bitsize - 1 - pos;
  }
  return result;
}
```

* using hardware provided AVX512 clz intrinsic (_mm(256|512)?_lzcnt_epi64)

Program your (custom) SIMD extension on FPGA

SIMD Types | SIMD Intrinsics | Custom SIMD Intrinsics

```
template<typename T, size_t VSizeBits>
inline auto clz(fpga_reg<T,VSizeBits> src) {
  fpga_reg<T,VSizeBits> result{};
  auto const bitsize = sizeof(T)*CHAR_BIT;
  using bitseq = ac_int<bitsize,false>;
  #pragma unroll
  for (auto i : indices(result)) {
    bitseq value(src[i]);
    int pos = bitsize - 1;
    #pragma unroll
    for(; pos>=0 && value[pos] == 0; pos--){};
    result[i] = bitsize - 1 - pos;
  }
  return result;
}
```



* using hardware provided AVX512 clz intrinsic (_mm(256|512)?_lzcnt_epi64)

Program your (custom) SIMD extension on FPGA

SIMD Types | SIMD Intrinsics | Custom SIMD Intrinsics

```
template<typename T, size_t VSizeBits>
inline auto hadd(fpga_reg<T,VSizeBits>::reg_t data) {
  T result;
  #pragma unroll
  for (auto i : indices(data)) {
    result += data[i];
  }
  return result;
}
```

Horizontal Operations

- "Melt" elements from register together (e.g., accumulate values from running-example)
- Strong data dependencies prevent proper pipelining

Program your (custom) SIMD extension on FPGA

SIMD Types | SIMD Intrinsics | Custom SIMD Intrinsics

```
template<typename T, size_t VSizeBits>
inline auto hadd(fpga_reg<T,VSizeBits>::reg_t data) {
  T result;
  #pragma unroll
  for (auto i : indices(data)) {
    result += data[i];
  }
  return result;
}
```

Horizontal Operations

- "Melt" elements from register together (e.g., accumulate values from running-example)
- Strong data dependencies prevent proper pipelining

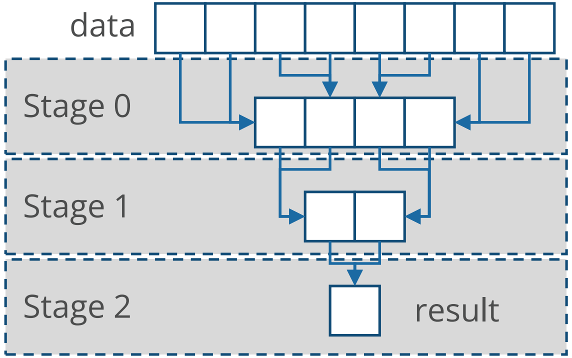
Solution: Divide-and-Conquer

Program your (custom) SIMD extension on FPGA

Dresden Database Research Group

SIMD Types | SIMD Intrinsics | Custom SIMD Intrinsics

```
template<typename T, size_t VSizeBits>
inline auto hadd(fpga_reg<T,VSizeBits>::reg_t data) {
  T result;
  #pragma unroll
  for (auto i : indices(data)) {
    result += data[i];
  }
  return result;
}
```



Horizontal Operations

- "Melt" elements from register together (e.g., accumulate values from running-example)
- Strong data dependencies prevent proper pipelining

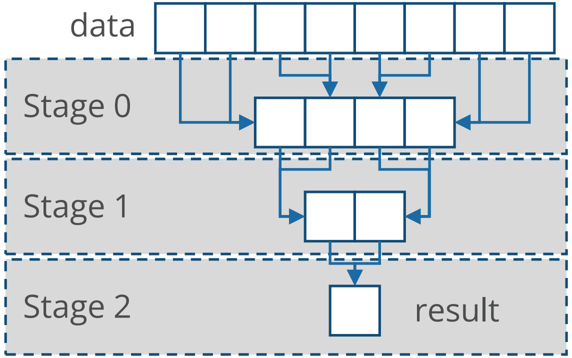
Solution: Divide-and-Conquer

- Recursive add of adjacent pairs

Program your (custom) SIMD extension on FPGA

SIMD Types | SIMD Intrinsics | Custom SIMD Intrinsics

```
template<typename T, size_t VSizeBits>
inline auto hadd(fpga_reg<T,VSizeBits>::reg_t data) {
  T result;
  #pragma unroll
  for (auto i : indices(data)) {
    result += data[i];
  }
  return result;
}
```



Horizontal Operations

- "Melt" elements from register together (e.g., accumulate values from running-example)
- Strong data dependencies prevent proper pipelining

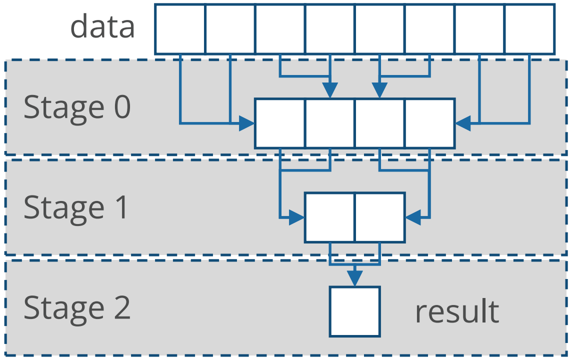
Solution: Divide-and-Conquer

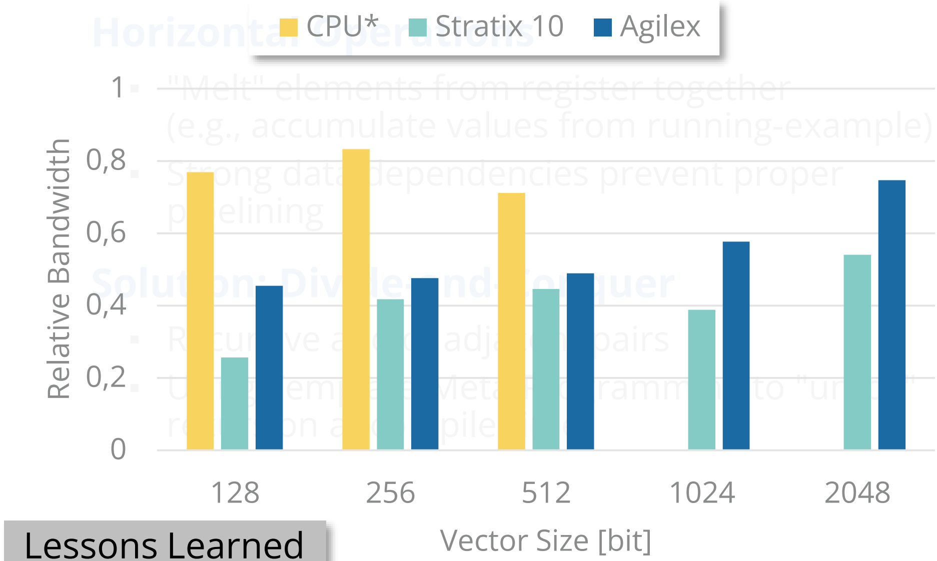
- Recursive add of adjacent pairs
- Using Template-Meta-Programming to "unroll" recursion at compile-time

Program your (custom) SIMD extension on FPGA

SIMD Types | SIMD Intrinsics | Custom SIMD Intrinsics

```
template<typename T, size_t VSizeBits>
inline auto hadd(fpga_reg<T,VSizeBits>::reg_t data) {
  T result;
  #pragma unroll
  for (auto i : indices(data)) {
    result += data[i];
  }
  return result;
}
```





Lessons Learned

- Data dependencies hurt the performance (still may outperform CPUs in absolute numbers)
- FPGAs require different processing strategies
- Hide complexity through custom intrinsics

* using hardware provided SSE / AVX(512) intrinsics

Excurse: SIMDified Binary Packing

Dresden Database Research Group

SIMD-BP on CPUs

- State-of-the-art integer compression schema
- Null-Suppression (eliminate unnecessary leading zero bits from data)

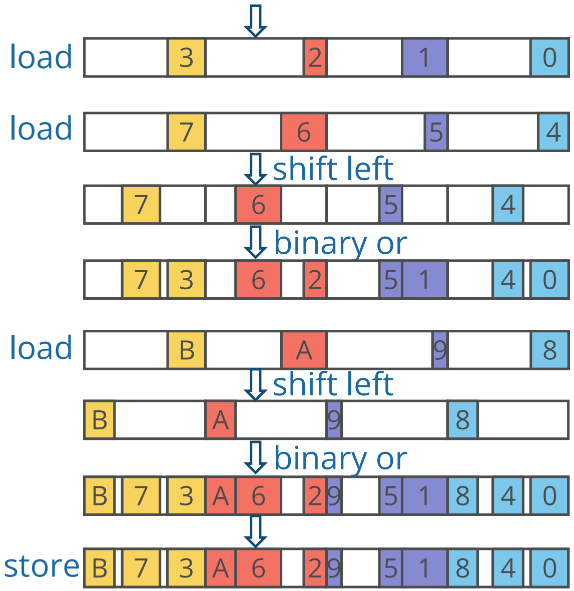
Core Ideas

- Encode a block of data elements with a fixed sized bitwidth (based on the maximum value)
- For linear memory access intertwine the data elements
- Fast (de-)compression through cheap instructions (element-wise logical shift and OR)

Drawbacks

- Block size is determined by SIMD register size
- Changed order of values in compressed result
- The bigger the block, the lower the compression rate

Example: Compressing with 24bits/int



Program your (custom) SIMD extension on FPGA

SIMD Types | SIMD Intrinsics | Custom SIMD Intrinsics

CPU

```
template<typename T>
void aggregate(auto out, auto in, size_t n) {
  auto const stepwidth = 8;
  __m512i result = _mm512_setzero_si512();
  auto const end = in+n;
  for (; in!=end; in+=stepwidth) {
    __m512i data = _mm512_loadu_si512(in);
    result = _mm512_add_epi64(result, data);
  }
  *out = _mm512_reduce_add_epi64(result);
}
```

Program your (custom) SIMD extension on FPGA

Dresden Database Research Group

SIMD Types | SIMD Intrinsics | Custom SIMD Intrinsics

CPU

```
template<typename T>
void aggregate(auto out, auto in, size_t n) {
  auto const stepwidth = 8;
  __m512i result = _mm512_setzero_si512();
  auto const end = in+n;
  for (; in!=end; in+=stepwidth) {
    __m512i data = _mm512_loadu_si512(in);
    result = _mm512_add_epi64(result, data);
  }
  *out = _mm512_reduce_add_epi64(result);
}
```

custom SIMD extension →

FPGA

```
template<typename T, size_t VSizeBits>
void aggregate(auto out, auto in, size_t n) {
  auto const stepwidth = intelFPGA<T,VSizeBits>::VL();
  auto result = set1<T,VSizeBits>(0);
  auto const end = in+n;
  for (; in!=end; in+=stepwidth) {
    auto data = load<T,VSizeBits>(in);
    result = add<T,VSizeBits>(result, data);
  }
  *out = reduce_add<T,VSizeBits>(result);
}
```

Program your (custom) SIMD extension on FPGA

SIMD Types | SIMD Intrinsics | **Custom SIMD Intrinsics**

CPU

```
template<typename T>
void aggregate(auto out, auto in, size_t n) {
  auto const stepwidth = 8;
  __m512i result = _mm512_setzero_si512();
  auto const end = in+n;
  for (; in!=end; in+=stepwidth) {
    __m512i data = _mm512_loadu_si512(in);
    result = _mm512_add_epi64(result, data);
  }
  *out = _mm512_reduce_add_epi64(result);
}
```

custom SIMD extension →

FPGA

```
template<typename T, size_t VSizeBits>
void aggregate(auto out, auto in, size_t n) {
  auto const stepwidth = intelFPGA<T,VSizeBits>::VL();
  auto result = set1<T,VSizeBits>(0);
  auto const end = in+n;
  for (; in!=end; in+=stepwidth) {
    auto data = load<T,VSizeBits>(in);
    result = add<T,VSizeBits>(result, data);
  }
  *out = reduce_add<T,VSizeBits>(result);
}
```

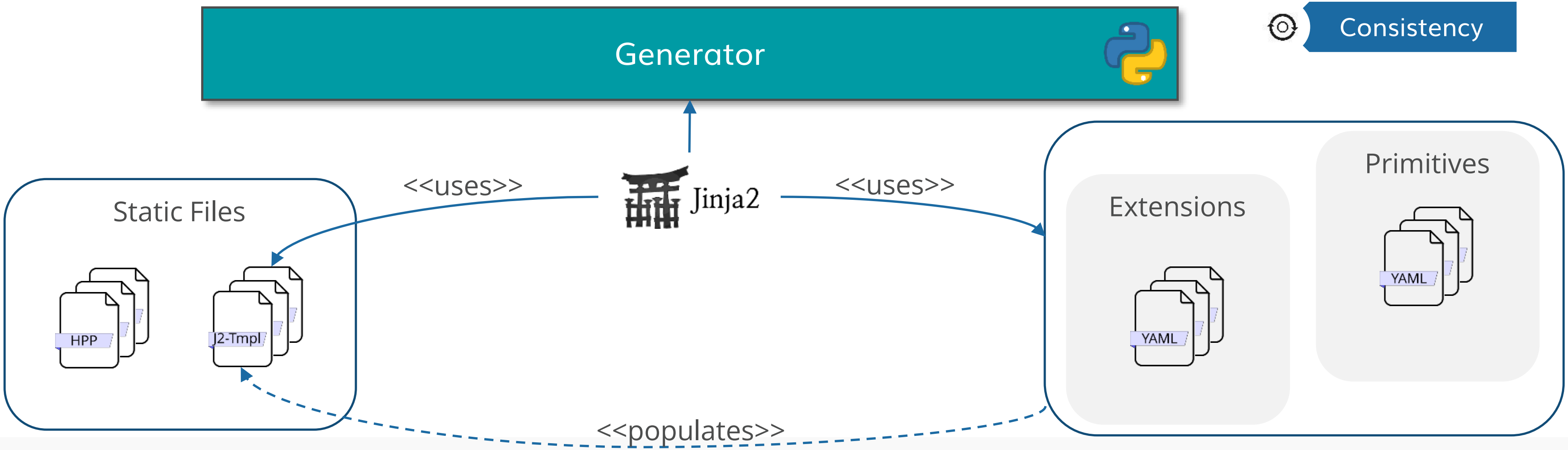
But there is more…

- Special arithmetic types
- Software-specific intrinsics written in C++ (HW/SW-codesign)
- Use-Case: Lightweight Compression Algorithm (included in the paper)

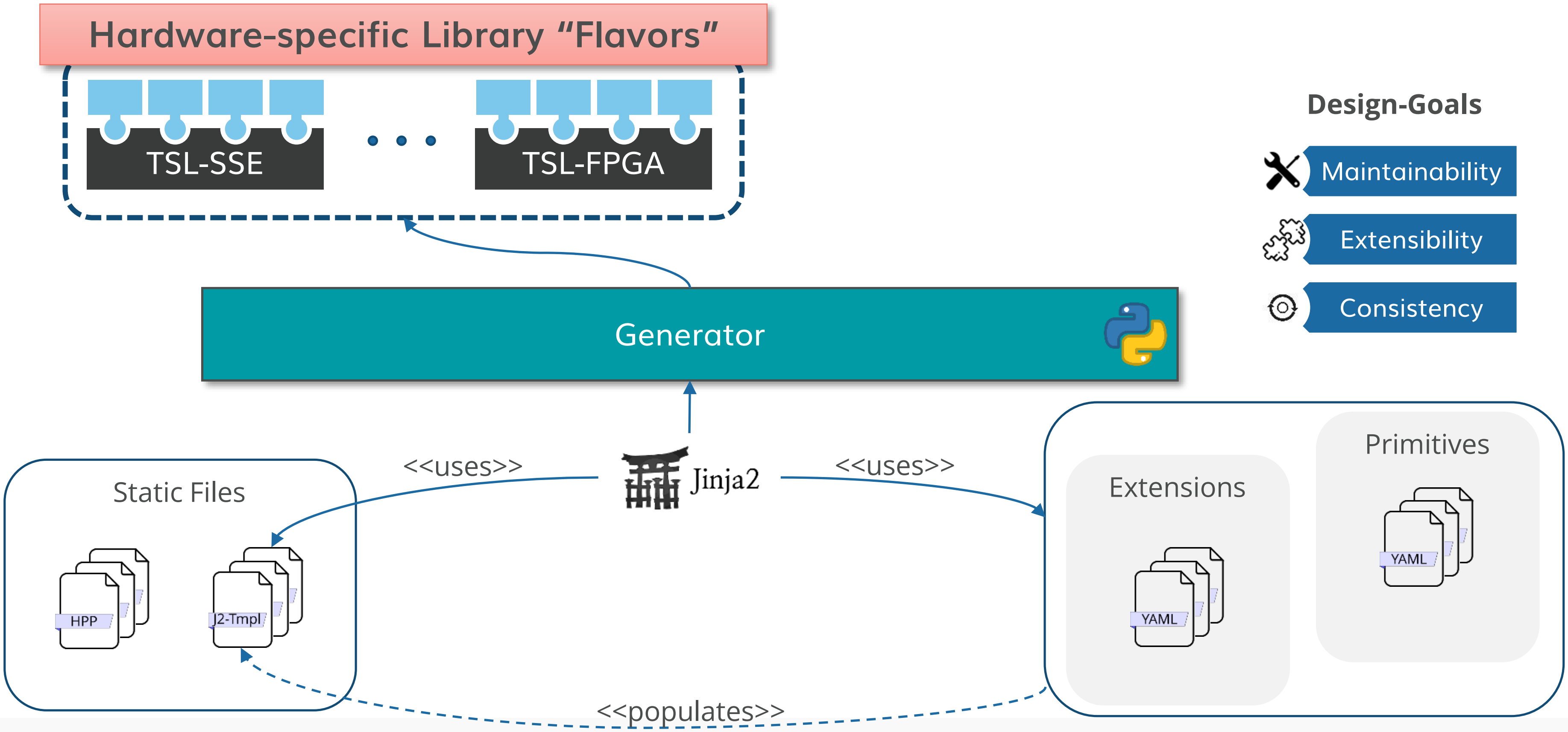
TSL – A Generator Based SIMD Abstraction Library

Design-Goals

- Maintainability
- Extensibility
- Consistency



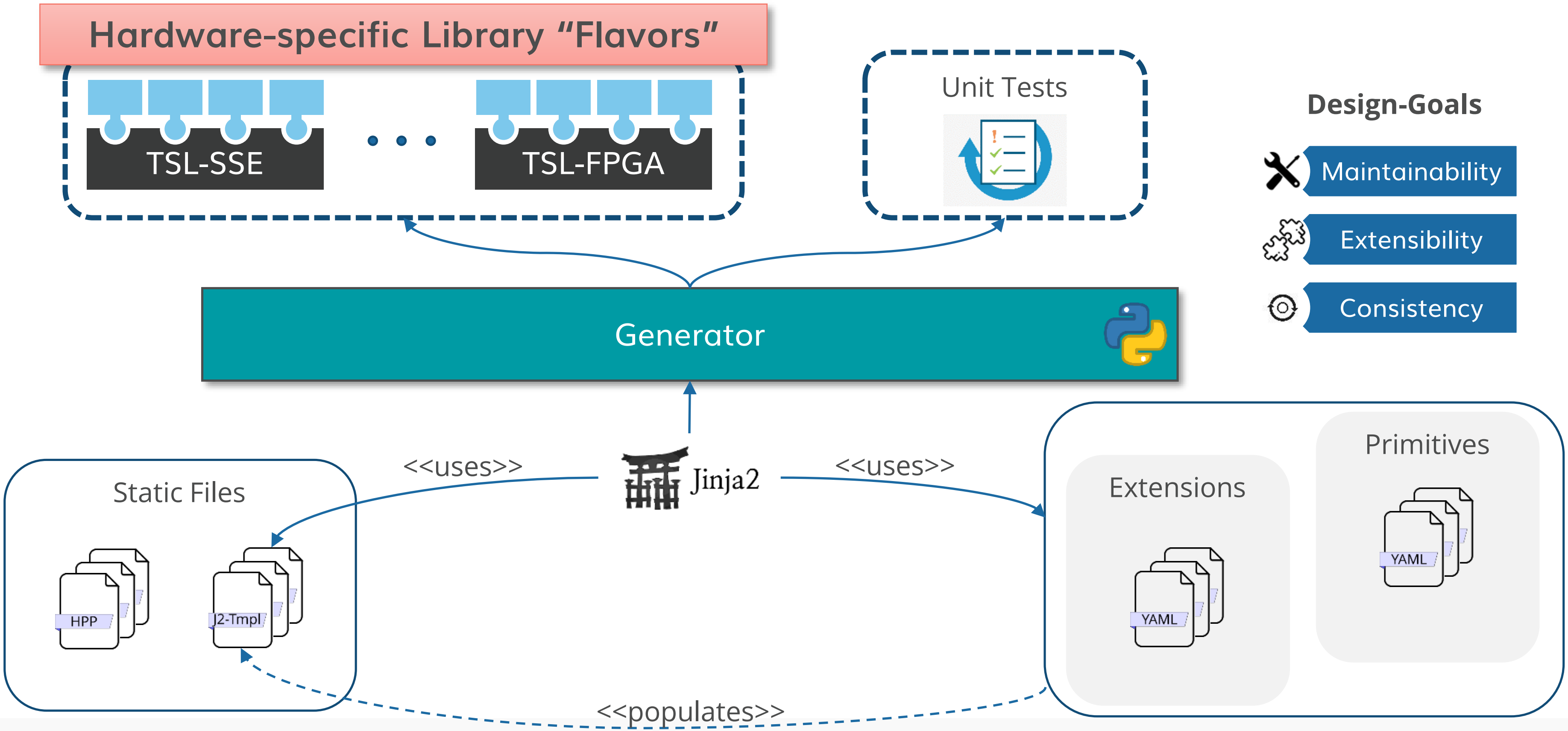
TSL – A Generator Based SIMD Abstraction Library



Design-Goals

- Maintainability
- Extensibility
- Consistency

TSL – A Generator Based SIMD Abstraction Library



Design-Goals

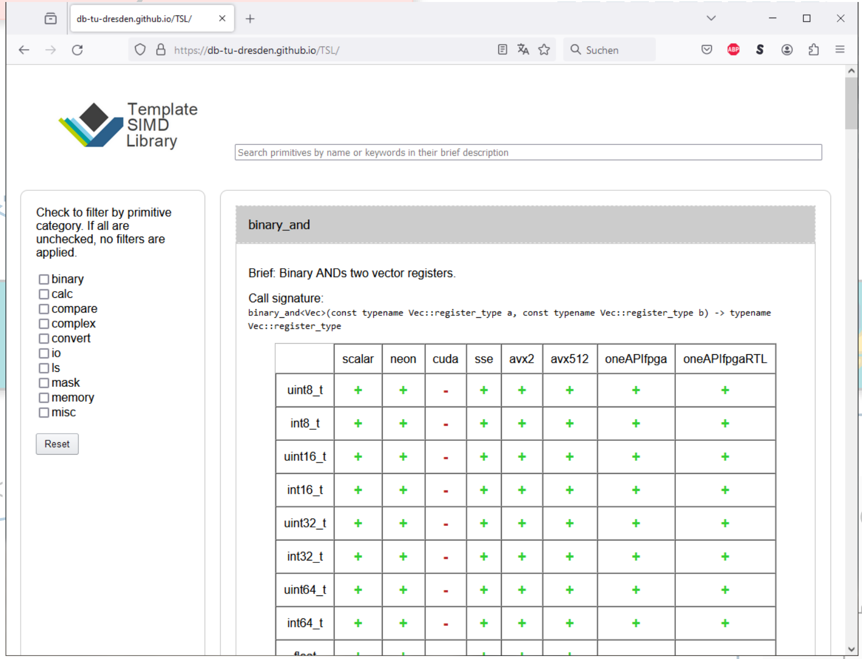
- Maintainability
- Extensibility
- Consistency

TSL – A Generator Based SIMD Abstraction Library

Dresden Database Research Group

Hardware-specific Library “Flavors”

TSL-SSE



db-tu-dresden.github.io/TSL/
https://db-tu-dresden.github.io/TSL/
Suchen

Template SIMD Library

Search primitives by name or keywords in their brief description

Check to filter by primitive category. If all are unchecked, no filters are applied.

- □ binary
- □ calc
- □ compare
- □ complex
- □ convert
- □ io
- □ ls
- □ mask
- □ memory
- □ misc

Reset

binary_and

Brief: Binary ANDs two vector registers.

Call signature:

```
binary_and<Vec>(const typename Vec::register_type a, const typename Vec::register_type b) -> typename Vec::register_type
```

	scalar	neon	cuda	sse	avx2	avx512	oneAPIfpga	oneAPIfpgaRTL
uint8_t	+	+	-	+	+	+	+	+
int8_t	+	+	-	+	+	+	+	+
uint16_t	+	+	-	+	+	+	+	+
int16_t	+	+	-	+	+	+	+	+
uint32_t	+	+	-	+	+	+	+	+
int32_t	+	+	-	+	+	+	+	+
uint64_t	+	+	-	+	+	+	+	+
int64_t	+	+	-	+	+	+	+	+

Design-Goals

- Maintainability
- Extensibility
- Consistency

Static Files

HPP

J2-Tmpl

ensions

Primitives

YAML

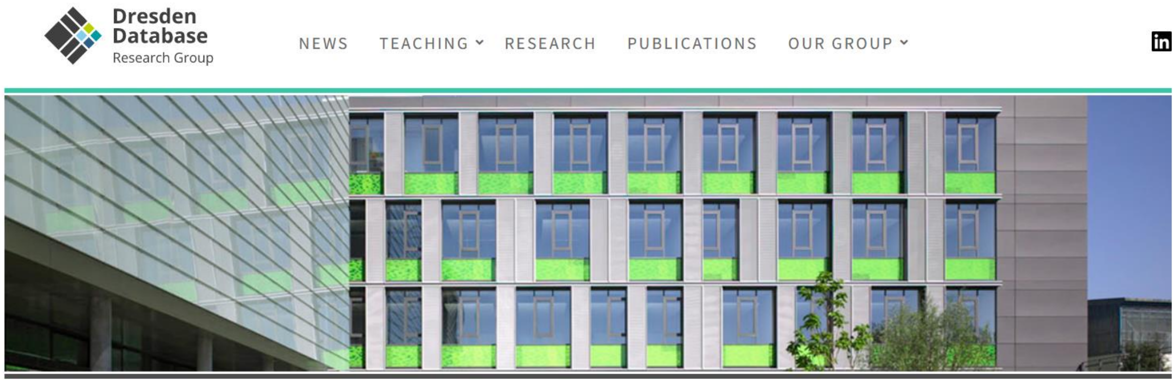
YAML

<<populates>>

Summary and Conclusion

Acknowledgements

The Dresden Database Team (esp. Axel, Rico, Lucas, Jerome, Dirk, Claudio, Alex, ...)



Sponsors

TECHNISCHE UNIVERSITÄT DRESDEN

DFG

SAP HANA

intel

NEC
NEC Technologies

Meta

Velox

GLOBAL FOUNDRIES

Conclusion and Outlook

THANKS to ... altera™ An Intel Company

FPGAs as accelerator in disaggregated computing environments ...

- ... are a given (and already widely available)
- ... come for free wrt host systems
- ... extremely beneficial for specific tasks

The good

- using HLS, FPGAs are "straight-forward" accessible by system-engineers
- exhaustive tool-support and existing frameworks further simplify the development
  - e.g., debugging, emulation, simulation
- state-of-the-art optimization techniques from CPU-world can be used to speed up FPGAs

The bad

- different general processing strategies compared to CPUs
- non-deterministic synthesizing (leading to variations in fmax → runtime)
- (very) long running synthesizing [1h, several days]