Technical Deep-Dive in a Column-Oriented In-Memory Database

Carsten Meyer, Martin Lorenz

carsten.meyer@hpi.de, martin.lorenz@hpi.de

Research Group of Prof. Hasso Plattner

Hasso Plattner Institute for Software Engineering
University of Potsdam

Goals

- Introduction to the technical foundations of a column-oriented, dictionary-encoded in-memory database
- Based on the “In-Memory Data Management” online lecture at openhpi.com

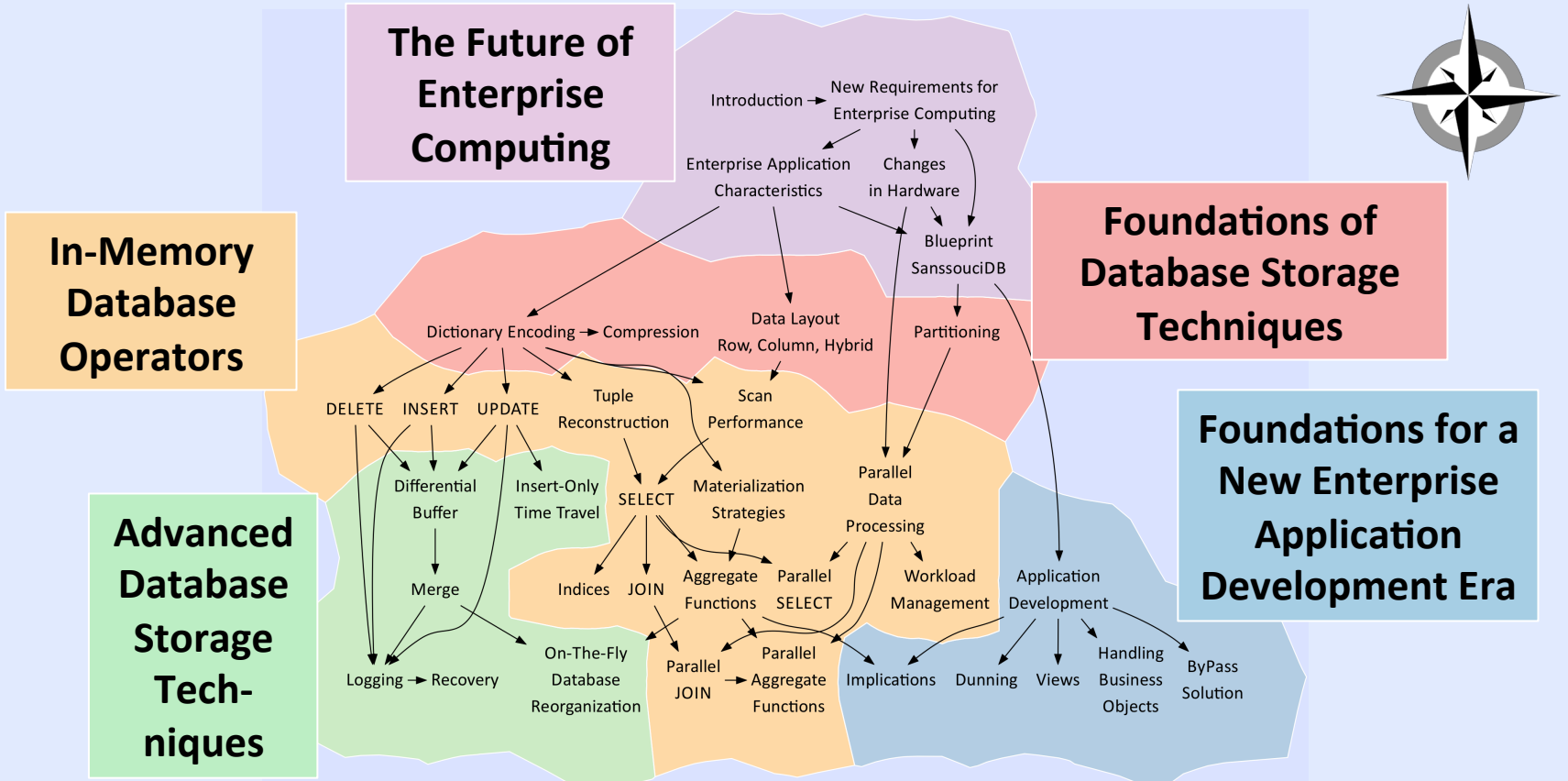
Chapters of the online course:

- 1 The future of enterprise computing
- **2 Foundations of database storage techniques**
- **3 In-memory database operators**
- 4 Advanced database storage techniques
- 5 Implications on Application Development

} In this talk

Learning Map of our Online Lecture @ openHPI.de

Hasso Plattner Institut
IT Systems Engineering | Universität Potsdam



HPI Hasso Plattner Institut
IT Systems Engineering | Universität Potsdam

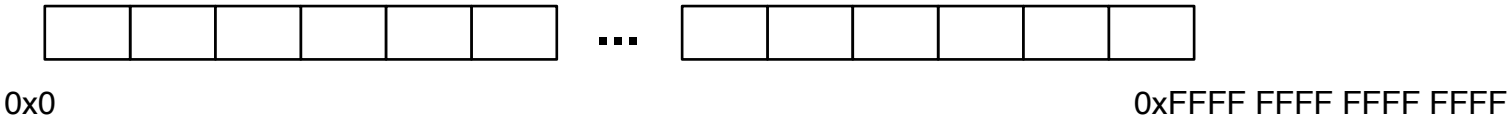
Chapter 2:

Foundations of Database Storage Techniques

HPI Hasso Plattner Institut
IT Systems Engineering | Universität Potsdam

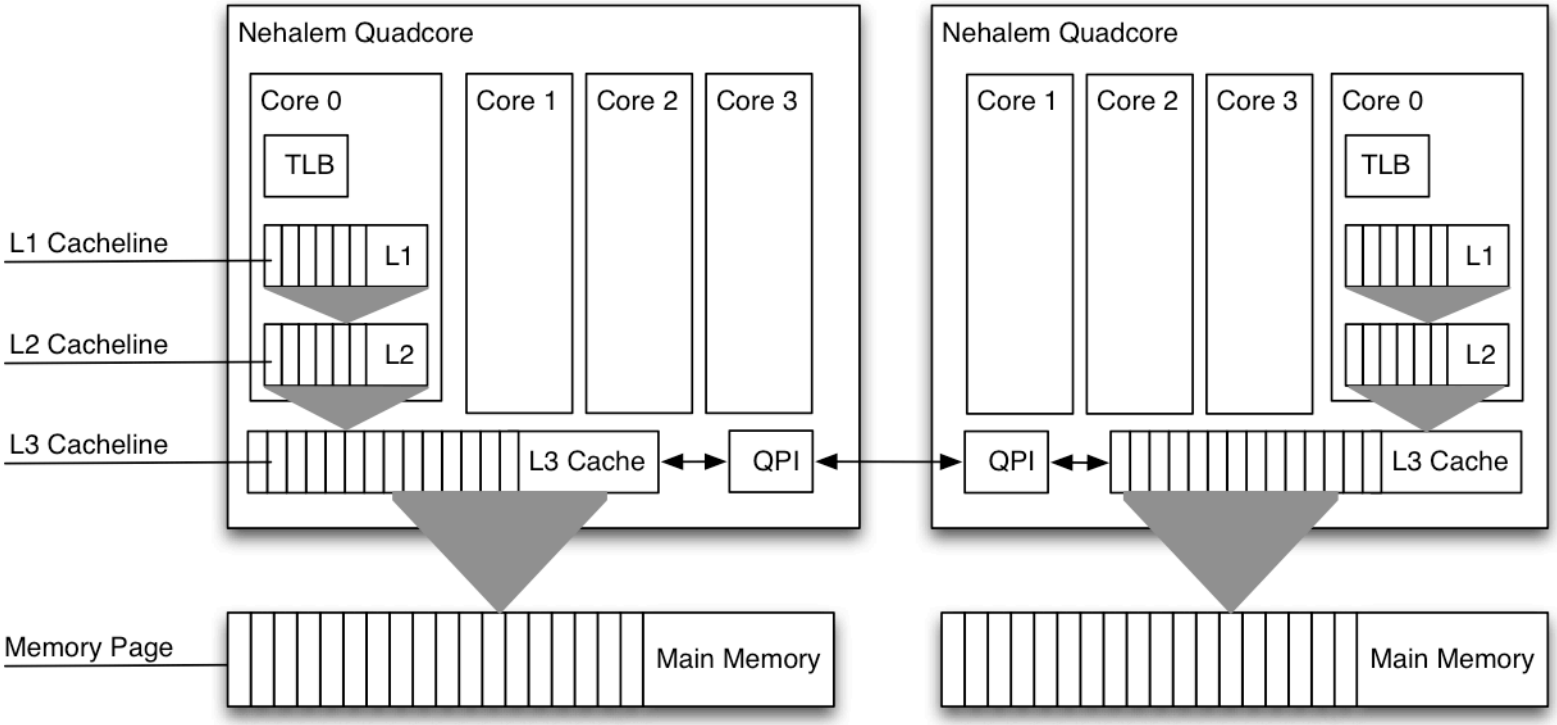
Data Layout in Main Memory

Memory Basics



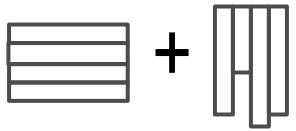
- Memory layout is **only linear**
- Every higher-dimensional access (like two-dimensional database tables) is mapped to this linear band

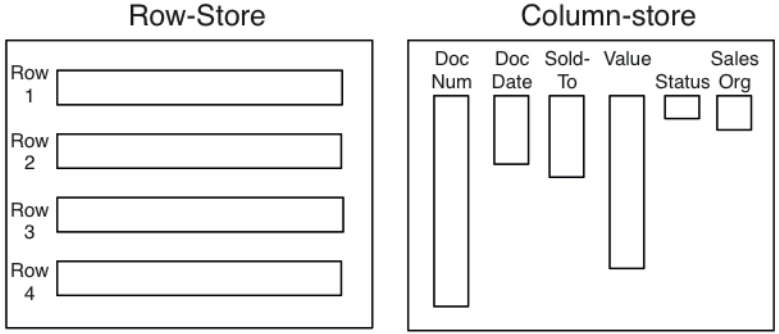
Memory Hierarchy



Physical Data Representation

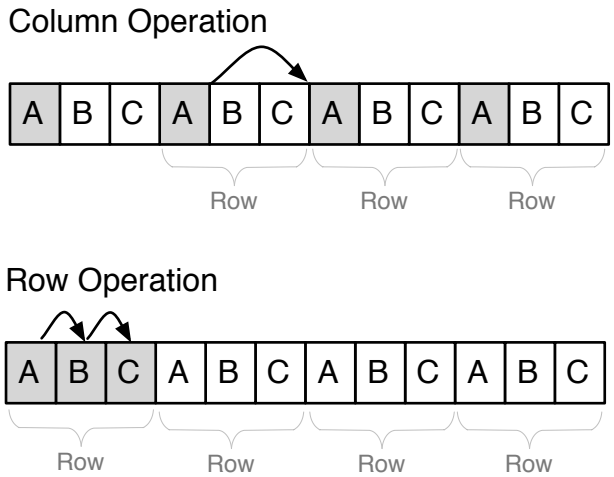
- **Row** store:
  - Rows are stored consecutively
  - Optimal for row-wise access (e.g. SELECT *)
- **Column** store:
  - Columns are stored consecutively
  - Optimal for attribute focused access (e.g. SUM, GROUP BY)
- Note: concept is **independent** from storage type





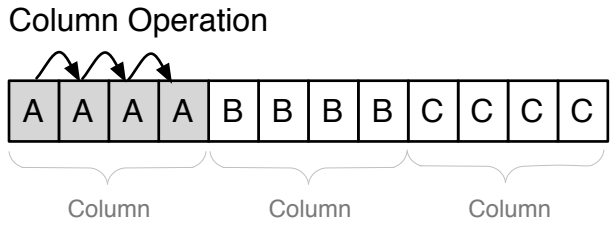
Row Data Layout

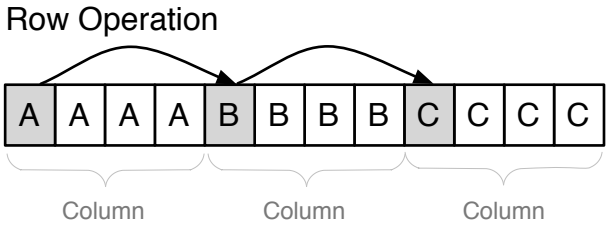
- Data is stored tuple-wise
- Leverage co-location of attributes for a single tuple
- Low cost for reconstruction, but higher cost for sequential scan of a single attribute



Columnar Data Layout

- Data is stored attribute-wise
- Leverage sequential scan-speed in main memory for predicate evaluation
- Tuple reconstruction is more expensive





Dictionary Encoding

Motivation

- Main memory access is the new bottleneck
- Idea: **Trade** CPU time to compress and decompress data
- Compression reduces number of memory accesses
- Leads to **less** cache misses due to more information on a cache line
- Operation directly on compressed data
- Offsetting with bit-encoded fixed-length data types
- Based on limited value domain

Dictionary Encoding Example

- **8 billion humans**

- Attributes:
  - first name
  - last name
  - gender
  - country
  - city
  - birthday

  ➔ 200 byte per tuple

- Each attribute is dictionary encoded

Sample Data

recID	fname	lname	gender	city	country	birthday
…	…	…	…	…	…	…
39	John	Smith	m	Chicago	USA	12.03.1964
40	Mary	Brown	f	London	UK	12.05.1964
41	Jane	Doe	f	Palo Alto	USA	23.04.1976
42	John	Doe	m	Palo Alto	USA	17.06.1952
43	Peter	Schmidt	m	Potsdam	GER	11.11.1975
…	…	…	…		…	…

Dictionary Encoding a Column

- A column is split into a dictionary and an attribute vector
- Dictionary stores all distinct values with implicit valueID
- Attribute vector stores valueIDs for all entries in the column
- Position is stored implicitly
- Enables offsetting with bit-encoded fixed-length data types

recID	fname
…	…
39	John
40	Mary
41	Jane
42	John
43	Peter
…	…

Dictionary for "fname"

valueID	Value
…	…
23	John
24	Mary
25	Jane
26	Peter
…	…

Attribute Vector for "fname"

position	valueID
…	…
39	23
40	24
41	25
42	23
43	26
…	…

Sorted Dictionary

- Dictionary entries are sorted either by their numeric value or lexicographically
  - → Dictionary lookup complexity: O(log(n)) instead of O(n)
- Dictionary entries can be compressed to reduce the amount of required storage
- Selection criteria with ranges are less expensive (order-preserving dictionary)

Data Size Examples

Column	Cardi-nality	Bits Needed	Item Size	Plain Size	Size with Dictionary (Dictionary + Column)	Compression Factor
First names	5 million	23 bit	50 Byte	373GB	238.4MB + 21.4GB	≈17
Last names	8million	23 bit	50 Byte	373GB	381.5MB + 21.4GB	≈17
Gender	2	1 bit	1 Byte	7GB	2.0b + 953.7MB	≈8
City	1million	20 bit	50 Byte	373GB	47.7MB + 18.6GB	≈20
Country	200	8 bit	47 Byte	350GB	9.2KB + 7.5GB	≈47
Birthday	40000	16 bit	2 Byte	15GB	78.1KB + 14.9GB	≈1
Totals			**200 Byte**	**≈ 1.6TB**	**≈ 92GB**	**≈17**

HPI Hasso Plattner Institut
IT Systems Engineering | Universität Potsdam

Chapter 3:

In-Memory Database Operators

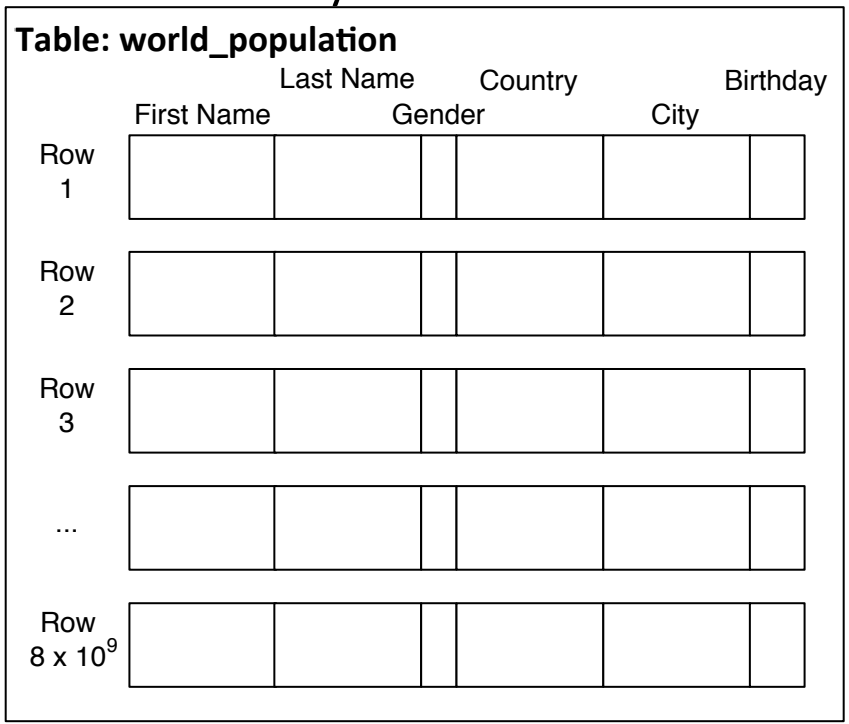
Scan Performance (1)

8 billion humans

- Attributes
  - First Name
  - Last Name
  - Gender
  - Country
  - City
  - Birthday
  - ➔ 200 byte
- Question: How many men/women?
- Assumed scan speed: 4MB/ms/core

Scan Performance (2)

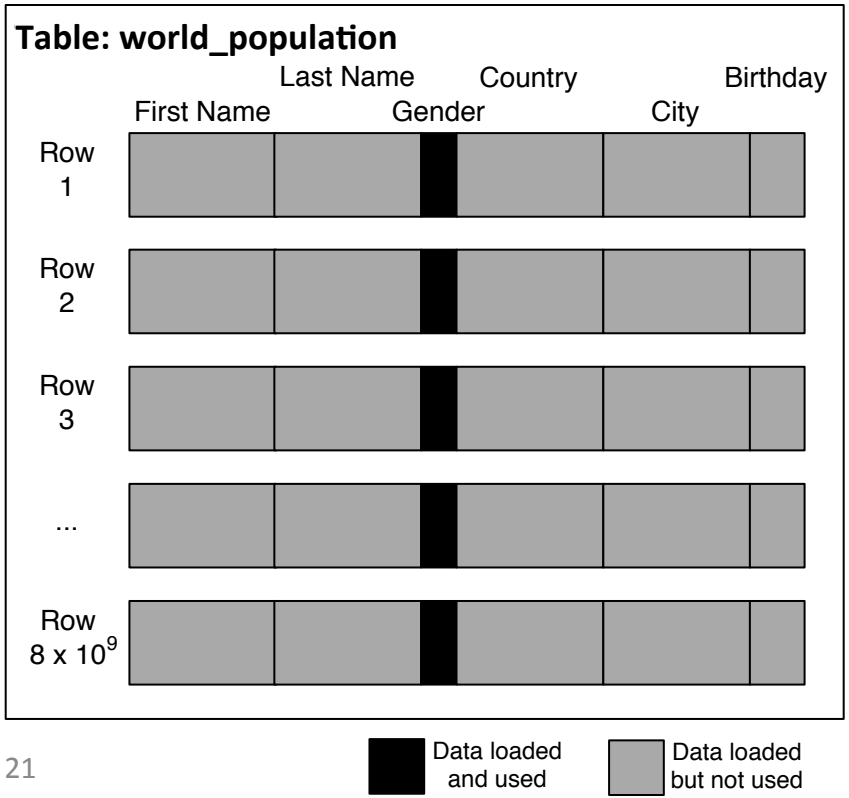
Row Store – Layout



- Table size:
  8 billion tuples × 200 bytes per tuple ≈ **1.6 TB**

Scan Performance (3)

Row Store – Full Table Scan

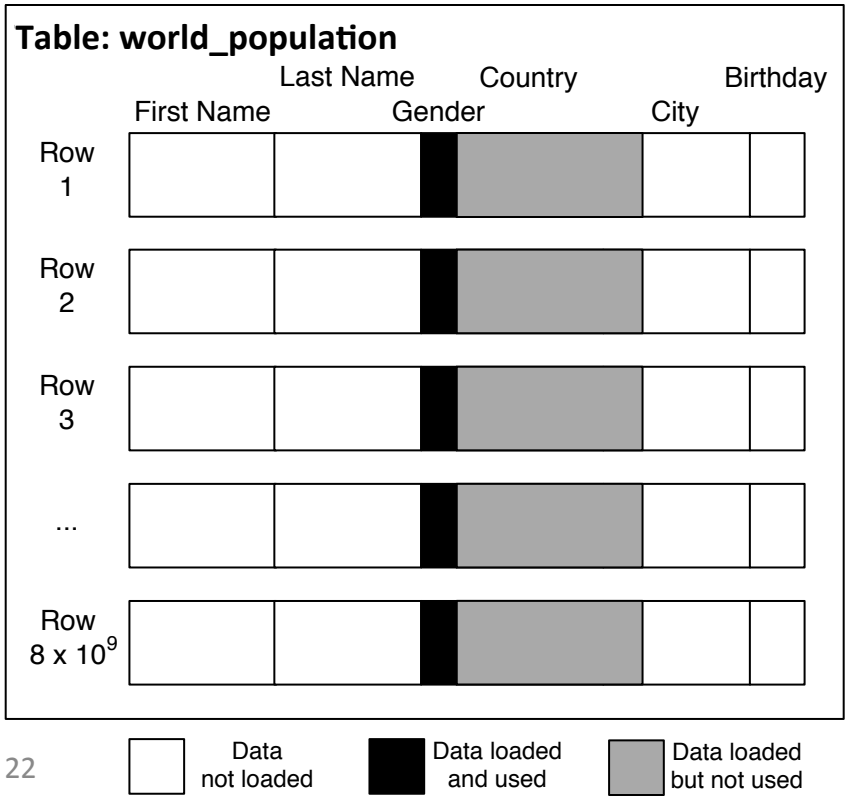


- Table size:
  8 billion tuples ×
  200 bytes per tuple
  ≈ 1.6 TB

- Scan through all rows with 4MB/ms/core
  → 400.000 ms

Scan Performance (4)

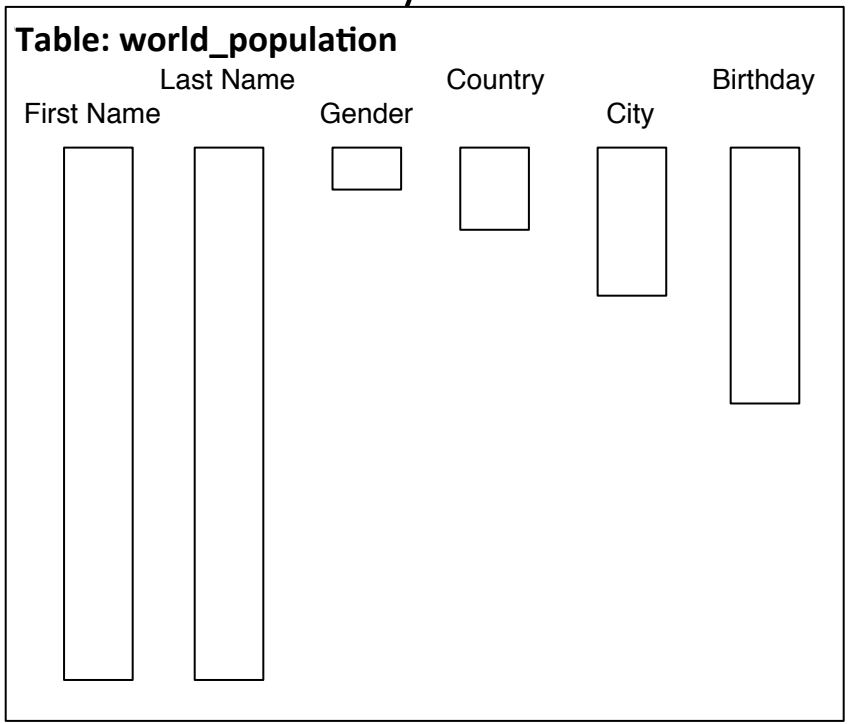
Row Store – Stride Access “Gender”



- 8 billion cache accesses à 64 byte ≈ **512 GB**
- Read with 4MB/ms/core **→ 128.000 ms**

Scan Performance (5)

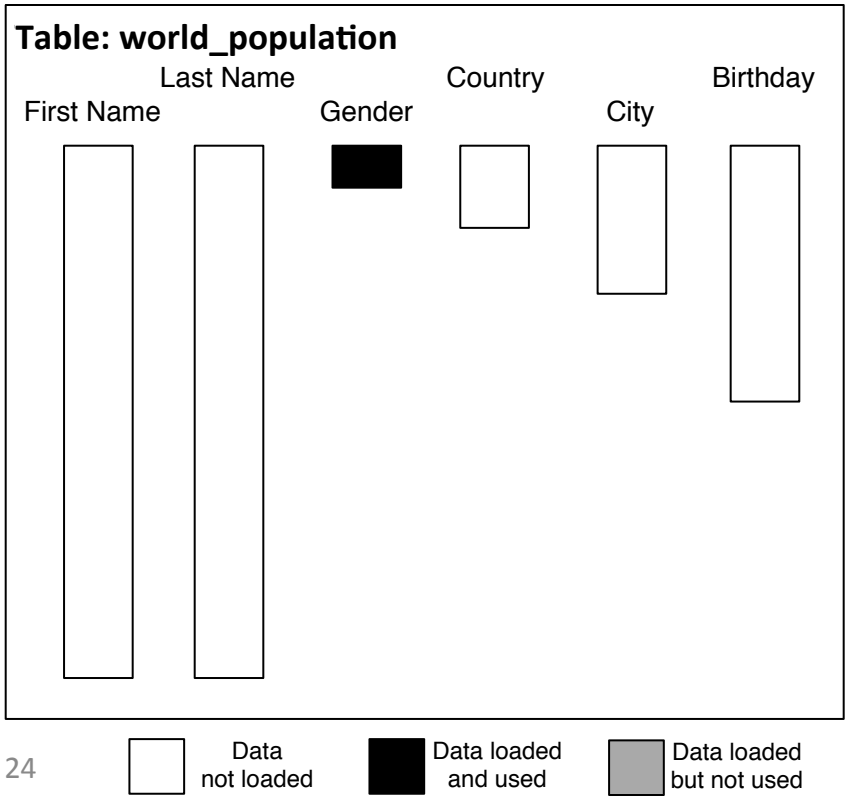
Column Store – Layout



- Attribute vector size ≈ **91 GB**
- Dictionary size ≈ **700 MB**
- Table size ≈ **92 GB**
- Compression factor ≈ **17**

Scan Performance (6)

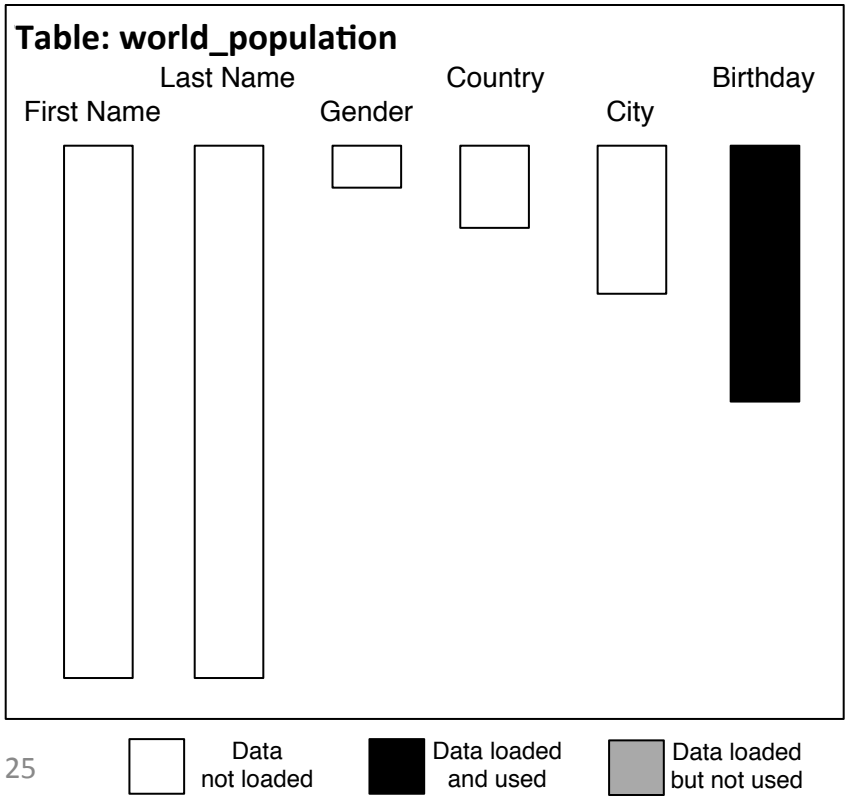
Column Store – Full Column Scan “Gender”



- Size of attribute vector “Gender”: 8 billion tuples × 1 bit per tuple
  ≈ 1 GB
- Scan through column with 4MB/ms/core
  → 250 ms

Scan Performance (7)

Column Store – Full Column Scan “Birthday”



- Size of attribute vector “Birthday”: 8 billion tuples × 2 byte per tuple
  ≈ 16 GB
- Scan through column with 4MB/ms/core
  → 4000 ms

Scan Performance – Summary

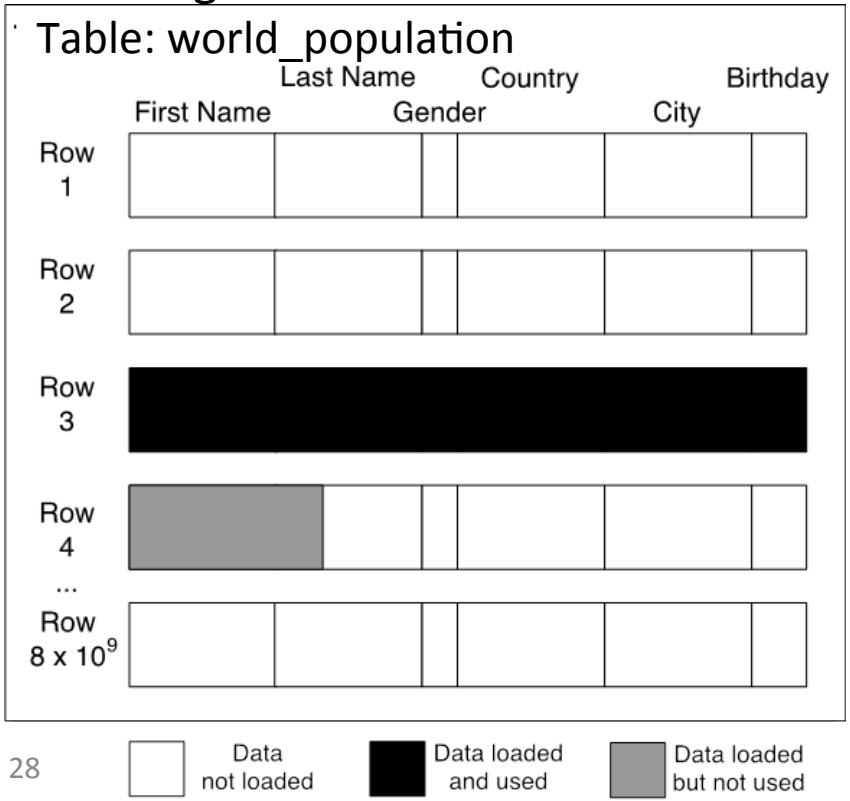
- How many women, how many men?

	Column Store	Row Store	
		Full table scan	Stride access
Time in seconds	250ms	400.000ms	128.000ms
		1,600x slower	512x slower

Tuple Reconstruction

Tuple Reconstruction (1)

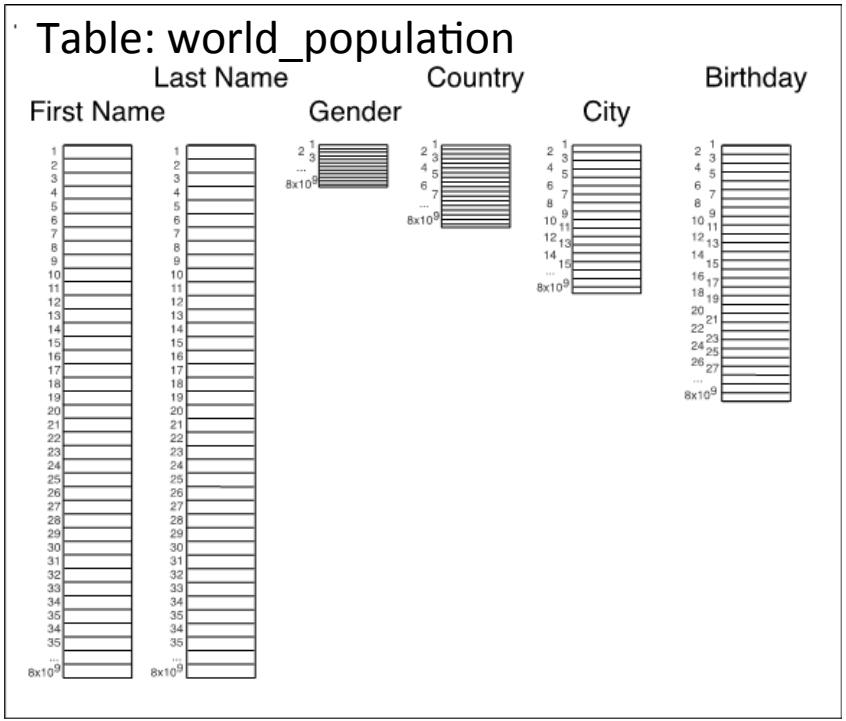
Accessing a record in a row store



- All attributes are stored consecutively
- 200 byte → 4 cache accesses à 64 byte → **256 byte**
- Read with 4MB/ms/core → ≈ **0.064 µs**

Tuple Reconstruction (2)

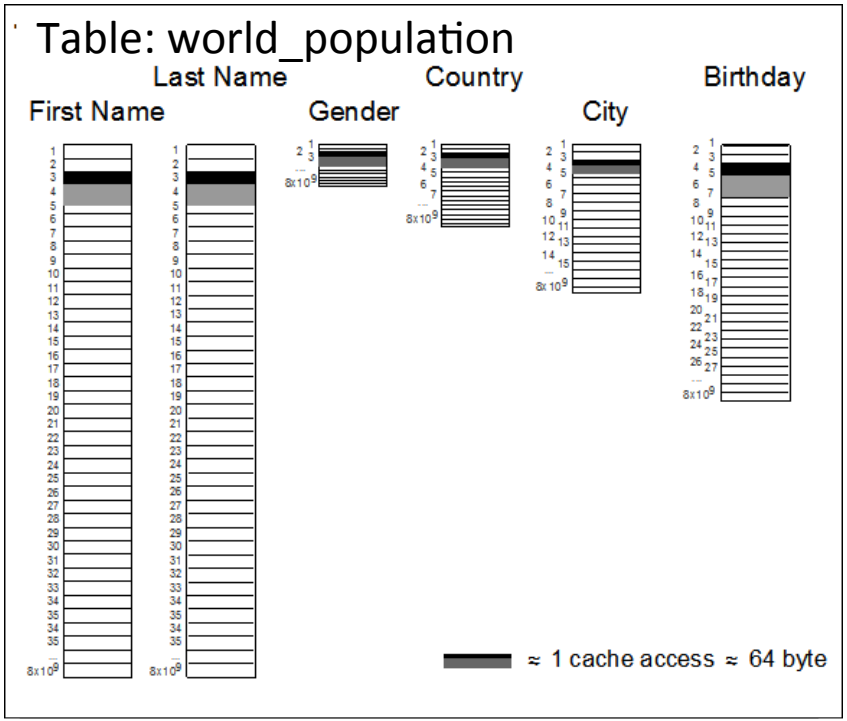
Virtual record IDs



- All attributes are stored in separate columns
- Implicit record IDs are used to reconstruct rows

Tuple Reconstruction (3)

Virtual record IDs



- 1 cache access for each attribute
- 6 cache accesses à 64 byte
  → **384 byte**
- Read with 4MB/ms/core
  → **≈ 0.093 µs**

Data not loaded | Data loaded and used | Data loaded but not used

Select

SELECT Example

```
SELECT fname, lname FROM world_population
WHERE country="Italy" and gender="m"
```

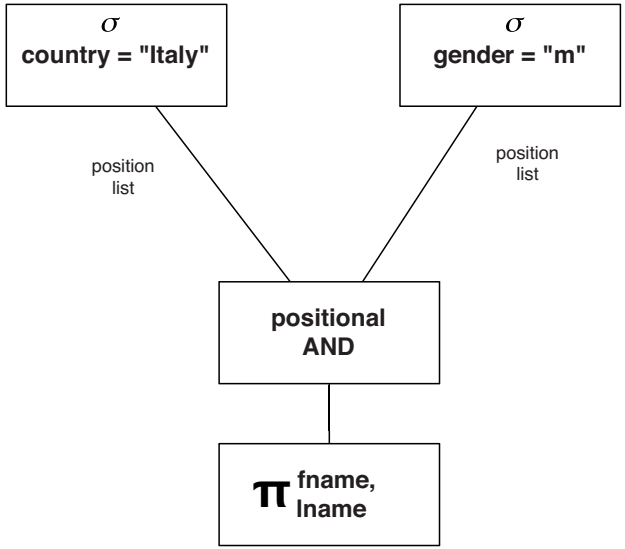
fname	lname	country	gender
Gianluigi	Buffon	Italy	m
Lena	Zaretti	Italy	f
Mario	Balotelli	Italy	m
Manuel	Neuer	Germany	m
Lukas	Podolski	Germany	m
Klaas-Jan	Huntelaar	Netherlands	m

Query Plan

- Multiple plans exist to execute query
  - Query Optimizer decides which is executed
  - Based on cost model, statistics and other parameters
- Alternatives
  - Scan “country” and “gender”, positional AND
  - Scan over “country” and probe into “gender”
  - Indices might be used
  - Decision depends on data and query parameters like e.g. selectivity

```
SELECT fname, lname FROM world_population
WHERE country="Italy" and gender="m"
```

Query Plan (i)



Positional AND:

- Predicates are evaluated and generate position lists
- Intermediate position lists are logically combined
- Final position list is used for materialization

Query Execution (i)

Value ID	Dictionary for "country"
0	Algeria
1	France
2	Germany
3	Italy
4	Netherlands
	…

fname	lname	country	gender
Gianluigi	Buffon	3	1
Lena	Zaretti	3	0
Mario	Balotelli	3	1
Manuel	Neuer	2	1
Lukas	Podolski	2	1
Klaas-Jan	Huntelaar	4	1

Value ID	Dictionary for "gender"
0	f
1	m

country = 3 ("Italy")

Position
0
1
2

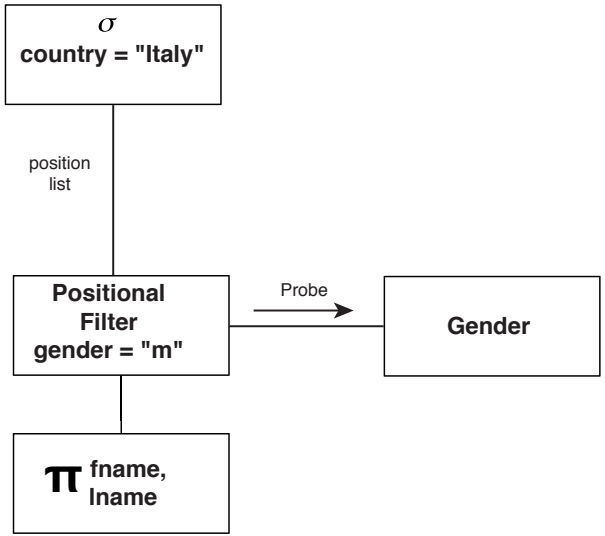
gender = 1 ("m")

Position
0
2
3
4
5

AND

Position
0
2

Query Plan (ii)



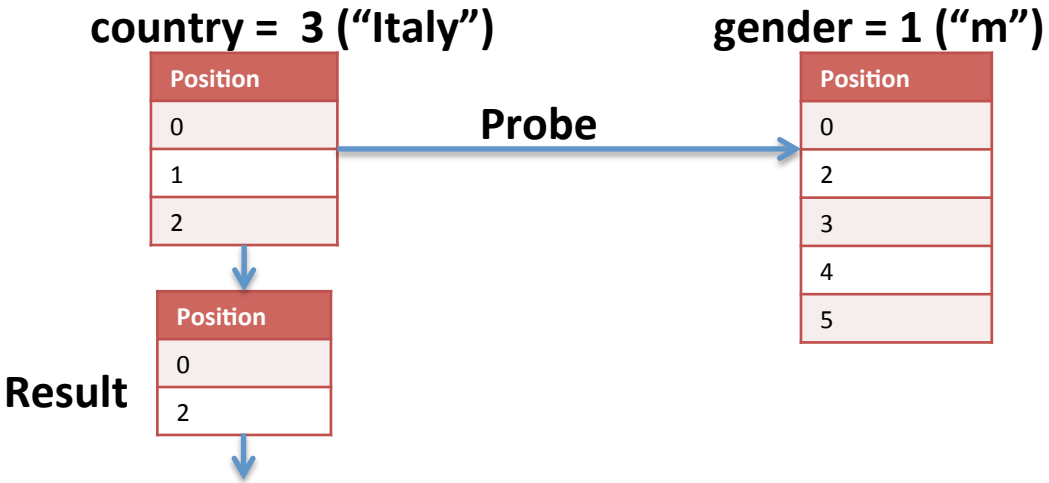
Based on position list produced by first selection, *gender* column is probed.

Query Execution (ii)

Value ID	Dictionary for "country"
0	Algeria
1	France
2	Germany
3	Italy
4	Netherlands
	…

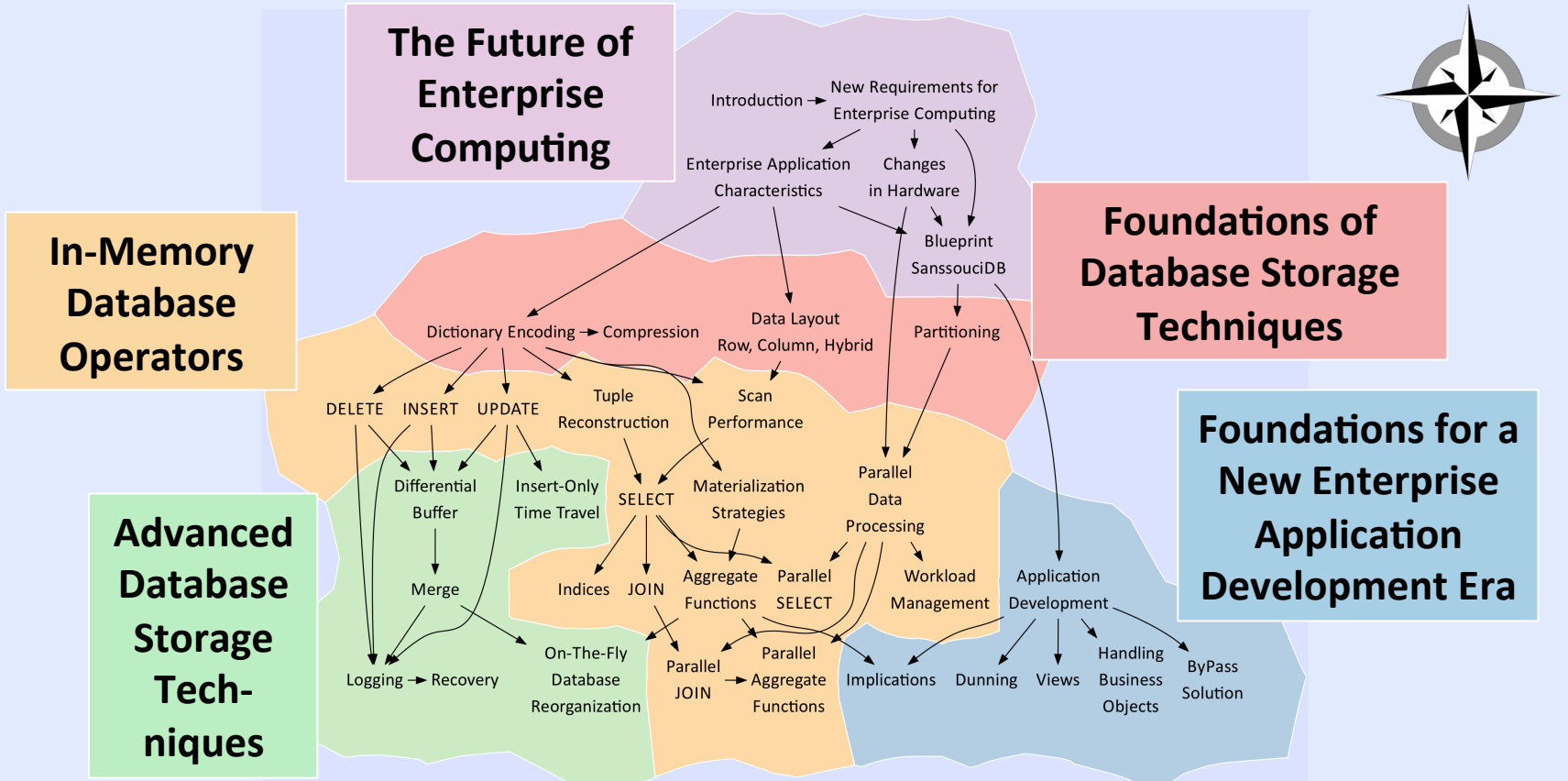
fname	lname	country	gender
Gianluigi	Buffon	3	1
Lena	Zaretti	3	0
Mario	Balotelli	3	1
Manuel	Neuer	2	1
Lukas	Podolski	2	1
Klaas-Jan	Huntelaar	4	1

Value ID	Dictionary for "gender"
0	f
1	m



Learning Map of our Online Lecture @ openHPI.de

Hasso Plattner Institut
IT Systems Engineering | Universität Potsdam



Come to our booth!

HPI Hasso Plattner Institut
IT Systems Engineering | Universität Potsdam

Location

#204 → In front of the keynote area

Contact Information

www.openhpi.com

Carsten.Meyer@hpi.de

Martin.Lorenz@hpi.de