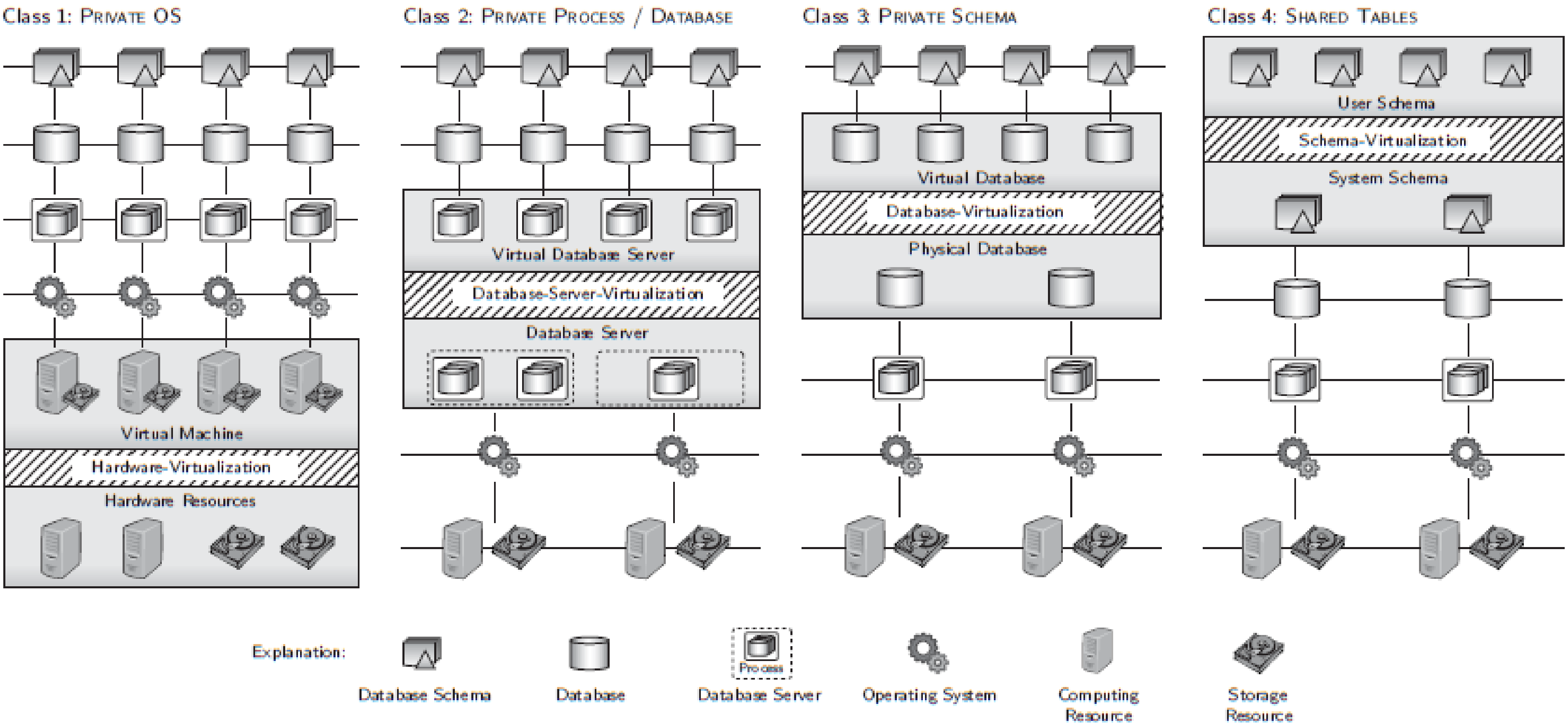
10 Multi-Tenancy Techniques

> Potential Virtualization Layers



> Virtualization Classes – Class 1

Class 1 (PRIVATE OS)

- physical virtualization occurs on the level of hardware resources (CPU, memory, network)
- Virtual machine monitors, e.g., VMware, XEN, or KVM
- each application owns a separate operating system as well as a database server and databases
- stronges t isolation level
- High resource usage per application -> poor utilization of the underlying hardware
- Typically only a few tens of different applications can be hosted on a single machine
- implementation transparently without any modification of the database management system

	Class 1	Class 2	Class 3	Class 4
Resources per Application (Costs)	– –	o	+ +	+ +
Resource Utilization / Scalability	– –	–	+	+ +
Provisioning Time and Costs	– –	–	+ +	+ +
Maintainability (Updates/Patches)	– –	–	+	+
Isolation (Performance)	+	+	o	–
Application Independence	+ +	+	+	–
Isolation (Security)	+ +	+ +	+	o
Maintainability (Backup/Restore)	+	+	o	– –

Class 2 (PRIVATE PROCESS / PRIVATE DATABASE)

- Logical virtualization occurs at the level of the database server
- Two slightly different ways
  - (1) Each virtual database server is executed in one or several private processes on the physical machine
  - (2) all virtual database servers are executed in a single server instance and each application creates private databases inside this server
- isolation is weaker compared to Class 1
- better scalability up to a few hundreds applications per machine
- Implementation requires the DBMS to either provide private server processes (e.g. as instances) or allow for private databases that are maintained and used by different applications without interfering with one another.

	Class 1	Class 2	Class 3	Class 4
Resources per Application (Costs)	– –	o	+ +	+ +
Resource Utilization / Scalability	– –	–	+	+ +
Provisioning Time and Costs	– –	–	+ +	+ +
Maintainability (Updates/Patches)	– –	–	+	+
Isolation (Performance)	+	+	o	–
Application Independence	+ +	+	+	–
Isolation (Security)	+ +	+ +	+	o
Maintainability (Backup/Restore)	+	+	o	– –

> Virtualization Classes – Class 3

Class 3 (PRIVATE SCHEMA)

- Each application accesses private tables and indexes that are in turn mapped to a single physical database
- different applications use the same physical database in a shared fashion
- weaker isolation compared to previous classes
  database components like buffer management, logging, etc. are shared
- Isolation with respect to security has to be enforced in the application or on the database level using access rights to different database objects (like tables).
- smaller footprint, good resource utilization, and scalability of up to a few thousand applications per machine
- Implementation requires modifications of today's database management systems in order to hide the existence of other users' database objects and activities from each application.

	Class 1	Class 2	Class 3	Class 4
Resources per Application (Costs)	– –	o	+ +	+ +
Resource Utilization / Scalability	– –	–	+	+ +
Provisioning Time and Costs	– –	–	+ +	+ +
Maintainability (Updates/Patches)	– –	–	+	+
Isolation (Performance)	+	+	o	–
Application Independence	+ +	+	+	–
Isolation (Security)	+ +	+ +	+	o
Maintainability (Backup/Restore)	+	+	o	– –

Class 4 (SHARED TABLES)

- Applications share all components in the software stack
- The database schema is virtualized such that each application sees a private schema; all private schemas are mapped to a single system schema
- best suited for Software-as-a-Service infrastructures when applications use the same or a very similar database schema ("Multi-Tenant-Databases")
- Weakness: poor performance or loss of strong typing
- offers the least isolation between applications
  - Security: enforced on application level or with row-level-authorization in the database system
  - no isolation with respect to performance or availability
- high complexity in the database management system – e.g., the query optimizer needs to be aware of the intermingled data – and complicates maintenance tasks like, e.g., backup, restore or migration of single tenants
- advantageous in using this class for extreme scalability to scale up to several thousands of applications per single machine.
- Major modifications necessary to support multi-tenancy with good performance

	Class 1	Class 2	Class 3	Class 4
Resources per Application (Costs)	– –	o	+ +	+ +
Resource Utilization / Scalability	– –	–	+	+ +
Provisioning Time and Costs	– –	–	+ +	+ +
Maintainability (Updates/Patches)	– –	–	+	+
Isolation (Performance)	+	+	o	–
Application Independence	+ +	+	+	–
Isolation (Security)	+ +	+ +	+	o
Maintainability (Backup/Restore)	+	+	o	– –

> Comparison

Criterion	Class 1	Class 2	Class 3	Class 4
Service Model	IaaS	PaaS	Paas, Saas?	SaaS -> DBaaS
Isolation	Highest	High	Moderate	Low
=> Data/Security	High (good)	High (good)	High (good) – auth based table access	Low (bad) – application
=> Performance/workload	High (good) – sep mem	High (good) – shared mem	Low (bad) – pooled mem	Low (bad) – shared objects
=> system/OS failure	High (good)	Moderate – OS☹ DBMS☺	Low – OS ☹, DBMS ☹	Low (bad)
=> Maintenance (backup/restore/migrate)	Simple migration	Separate backup/restore	Hard	Hard
=> Maintenance (patches)	High (bad)	Moderate – OS☺ DBMS☹	Low – OS ☺, DBMS ☺	Low (good)
=> Shared data access	High (bad)	High (bad)	Moderate (good)	Moderate (good)
Scalability	Few users per machine	Tens	Hundreds	Thousands
=> Price per user (footprint)	Highest (even when not active)	High (running instance consumes resources)	Low (inactive users do not consume resources)	Least
=> HA, failover costs -> replication	Highest	High	Moderate	Moderate
Sharing	Hardware	OS, DBMS files	Per DB: logging, backup, TS, BP, locks, catalog	Per table: statistics, compression, indexes, …
Utilization	Low	Moderate	High	Highest
Performance	Depends on #tenants per machine… probably getting worse towards the right (because of more #tenants)			
Implementation costs	Low (VMs)	Low (e.g., DB2 instances)	High (e.g., SQL Azure)	High (Mapping layer?)

> What is Multi-tenancy?

Multi-tenancy refers to the ability

- To run **multiple customers** on a **single software instance** installed on multiple servers
- This is done to **increase resource utilization** by allowing load balancing among tenants, and to
- **Reduce operational complexity and cost** in managing the software to deliver the service

From a customer's perspective

- Multi-tenancy is **transparent** → customer seems to have an instance of the software entirely to themselves
- **Customization** can be employed to the degree the application supports it without regard to what other tenants are doing

> What is Multi-tenancy? (2)

Consolidate multiple businesses (tenants) onto the same operational system

Pool resources to improve their utilization

- Avoid provisioning each tenant for their maximum load
- Breaks down isolation: weakens security, increases resource contention, interferes with optimizations

Provide a tenant-aware administrative framework to improve management efficiency

- Manage farms of individual multi-tenant servers
- Support bulk operations such as rolling upgrade
- Support tenant migration within and across farms

In this lecture: Focus on schemas and queries

> Multi-tenancy – literally

Multiple clients hosted by one service provider

- Multiple tenants hosted in one building complex

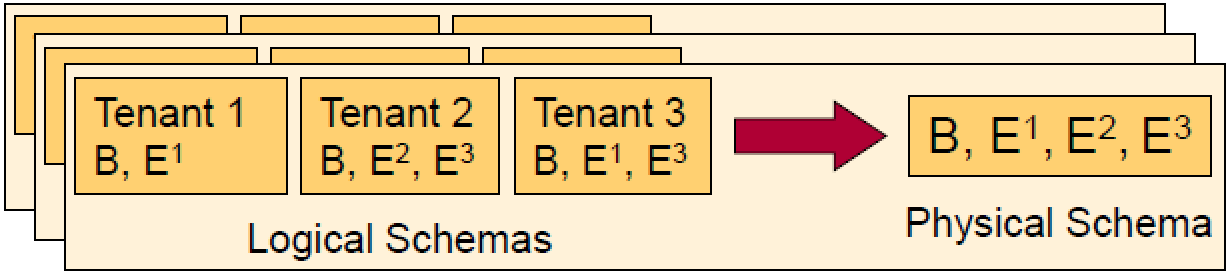
Code (to be executed)

- Utilities (gas, electric, water, waste)

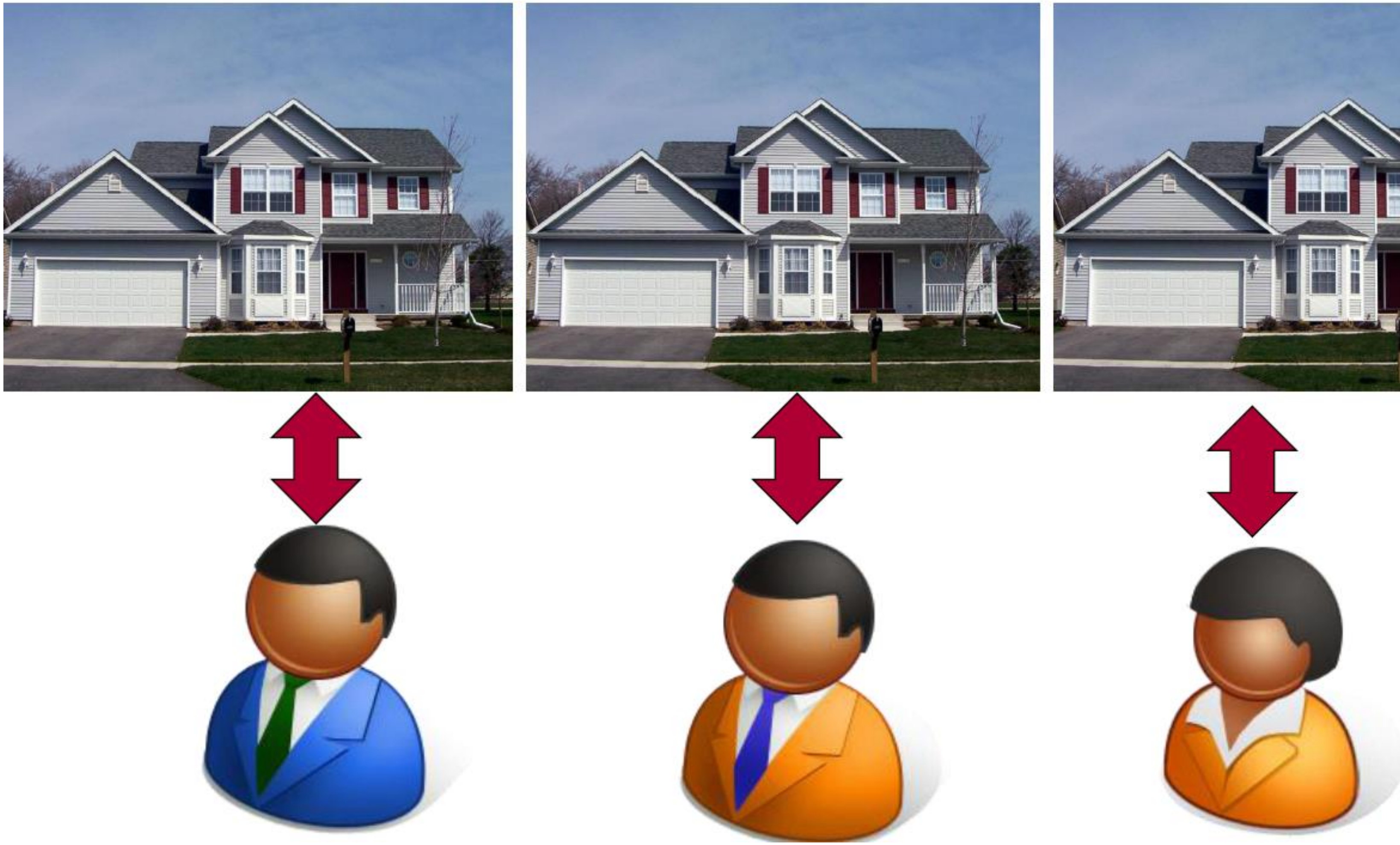
Data (with services)

- Storage space (furniture, basement, garage)

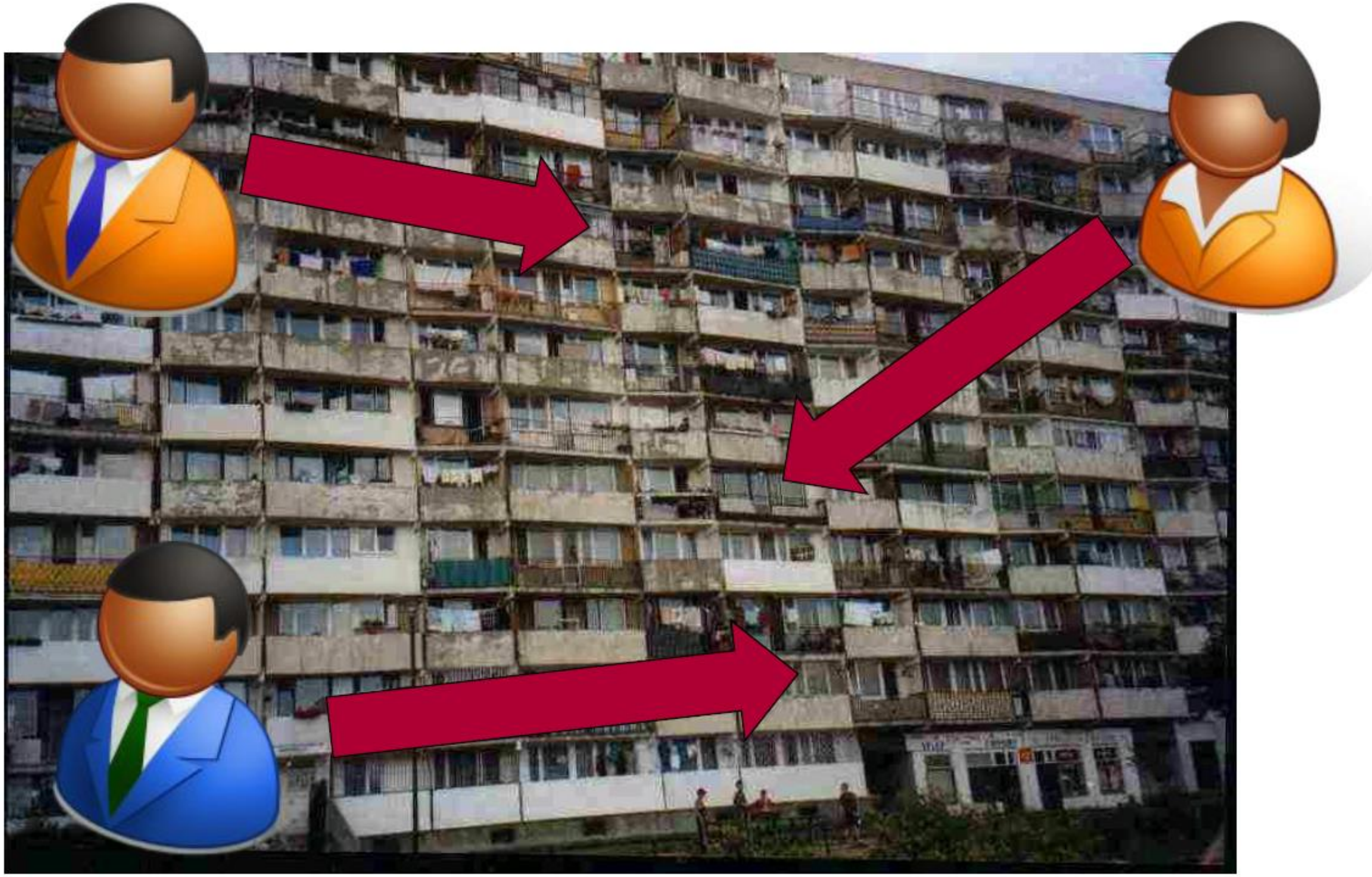
Four models…



> Single Homes

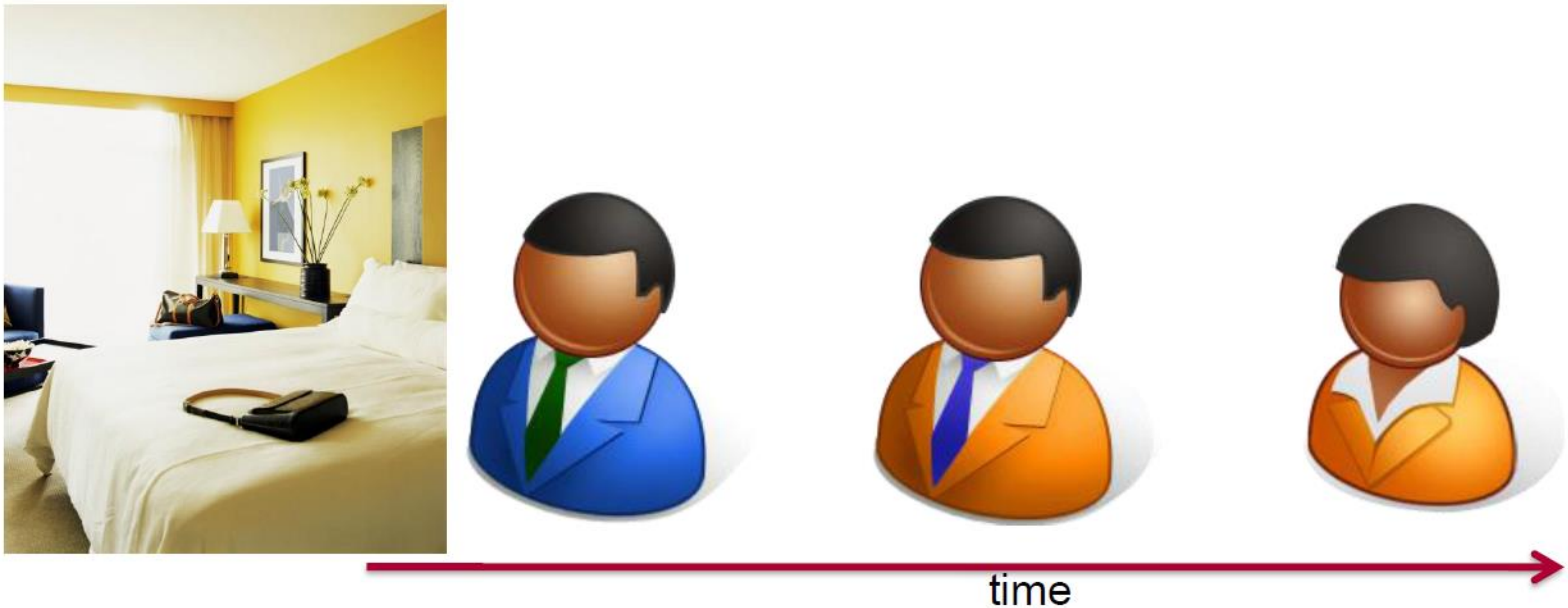


Different spaces, same time



Different spaces, same time, shared services

> Hotel Rooms



Same space, different times

> Youth Hostel

Same space, same time

Cost per person per night?

273 USD 54 USD 27 USD 10 USD

1,000,000 / 10 years / 365 day

Beds per m²?

6/500 = 0.012 4/100 = 0.04 2/25 = 0.08 6/25 = 0.24

> Isolation: Make other tenants invisible

High Fences

Strong Walls

Good Housekeeping

> Make other tenants invisible?

But here?

> Trust & Security



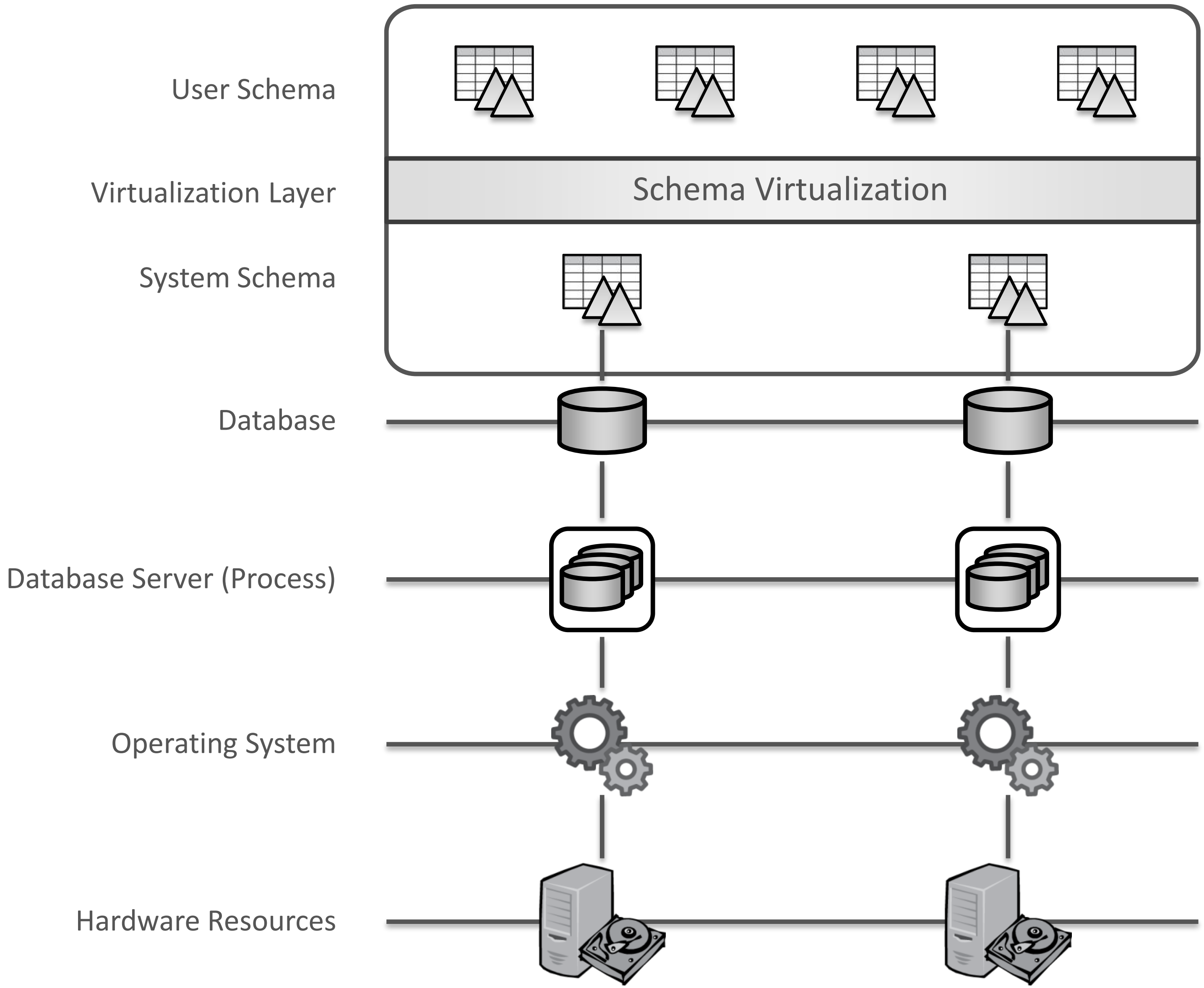

very high

high

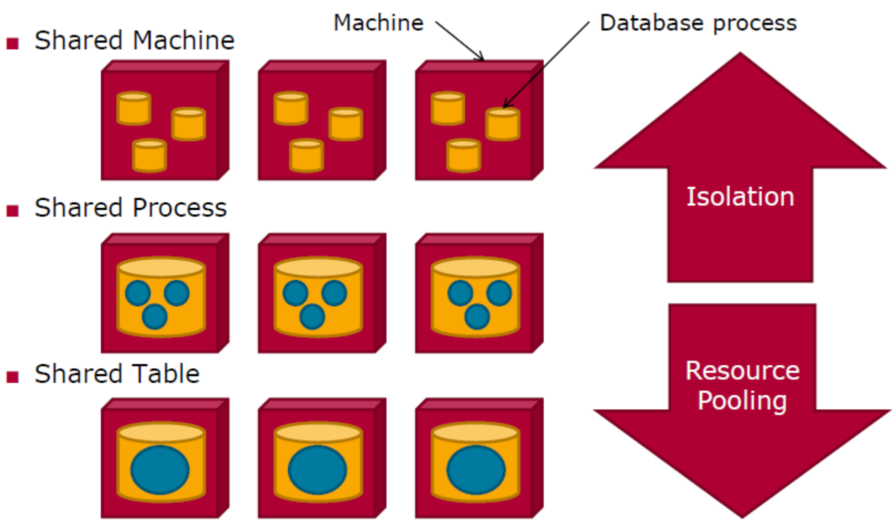
medium

low

> Class 4 Virtualization



Multitenancy Trade-offs

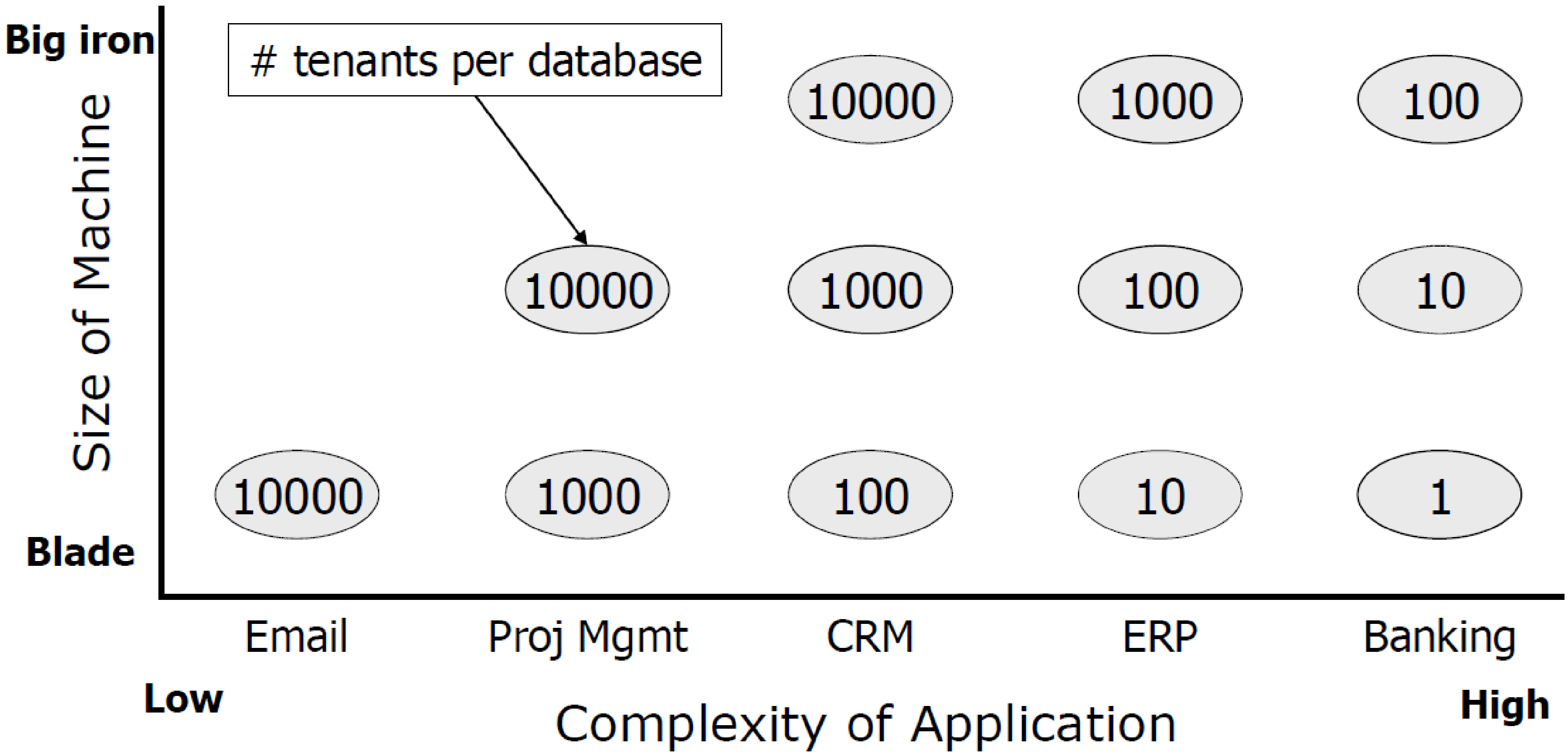


	Private Databases	Private Schema	Multi-tenancy
Simplicity	simple	simple (but need naming and mapping schemes)	hard
Customizability (schema)	high	high	low
Isolation	best	moderate	lowest
Resource Cost/tenant	high	low	lowest
#Tenants	low	large	largest

> Multitenancy Trade-offs (2)

	Private Databases	Private Schema	Multi-tenancy
Tools	Tools to deal with large number of DBs	Tools to deal with large number of tables	n/a
DB implementation cost	Lowest (query routing and simple mapping layer)	Low (query routing, simple mapping layer and query mapping)	High (query routing, simple mapping layer, query mapping, row-level isolation)
Scalability	Per tenant	Need some data/load balancing with dynamic migration	Need some data/load balancing with dynamic migration
Query Optimization	Less critical	Less critical	Critical (wrong plan over very large tables is disastrous)
Per Tenant Query Performance	As usual	Need query governance	Need query governance and tenant-specific statistics

> Multi-Tenancy in Practice



Aulbach et al.: Multi-Tenant Databases for Software as a Service: Schema-Mapping Techniques, SIGMOD 2008.

> Shared Machine / Private database

- Each customer gets his own database
- Main-Memory requirements for **one** database with **one** empty CRM schema instance

PostgreSQL	MaxDB	System X	System Y	System Z
55 MB	80 MB	171 MB	74 MB	273 MB

- Cannot scale beyond tens of tenants per server (→2007)
- Appropriate for applications with a smaller number of larger tenants, e.g., for banking

Jacobs, Dean and Aulbach, Stefan: Ruminations on Multi-Tenant Databases, BTW 2007

> Shared Process / Private schema

- Each customer gets their own tables within the same instance
- Main-Memory requirements for **one** database with **10,000** empty CRM schema instances

PostgreSQL	MaxDB	System X	System Y	System Z
79 MB	80 MB	616 MB	2061 MB	359 MB
55 MB	80 MB	171 MB	74 MB	273 MB

- Should scale up to thousands of tenants
- If each tenant gets their own table space, migration entails simply moving files
- Connection pooling is possible, but then tenant identity must be managed by the application

> Shared Table / Multi-tenancy

Data from many tenants in the same tables

- Add a **tenant_id** column
- Tenant queries must fix the value for this column
  - By connection or by application

Extend base schema using generic columns

- May be VARCHAR or a mix of types
- The database must compactly represent sparse tables

Advantage - everything is pooled

- Processes, memory, connections, prepared statements
- Easy DML and DDL operations across tenants
- Add, remove, and extend tenants with DML (not DDL)

Disadvantage - Isolation is very weak

- Irrelevant data infects query processing
  - Optimization statistics
  - Table scans
  - Data locality
  - No indexes or integrity constraints on generic columns
- Migration requires querying the operational system

Sharing

> Levels of Sharing

Tenant	Tuple	Pattern
1	N	Common Metadata
M	N	Locally Shared Data
*	N	Globally Shared Data
*	*	Common Data

Common Metadata

- Tenants share the definition of a table
- Content of the same tenant tables are stored in a single system table
- Tuples are associated with the tenant

Locally Shared Data

- Tenants additionally share tuples
- Shared tuples are accessed by groups of tenants
- Additional access table contains of tenant id and primary of the shared table

Globally Shared Data

- Shared tuples are always shared among all tenants
- Private tuples associated with a single tenant
- Globally shared tuples reference a special system tenant

Common Data

- All tenants share a complete table
- Useful for topographical entities such as cities, nations or currencies

> Common Metadata

Tenant 1

PRODUCT		
Product	Name	…
1	Camera 5I	
2	Camera 5II	

Tenant 2

PRODUCT		
Product	Name	…
1	TV 6000	

Tenant 3

PRODUCT		
Product	Name	…
1	Notebook X7	

Virtual Schemas

$\sigma_{\text{Tenant}=t}$

System Schema

PRODUCT			
Tenant	Product	Name	…
1	1	Camera 5I	
2	1	TV 6000	
1	2	Camera 5II	
3	1	Notebook X7	

> Locally Shared Data

Tenant 1

PRODUCT		
Product	Name	…
1	Camera 5I	
3	Camera 5II	

Tenant 2

PRODUCT		
Product	Name	…
2	TV 6000	

Tenant 3

PRODUCT		
Product	Name	…
2	TV 6000	
4	Notebook X7	

Virtual Schemas

$\sigma_{\text{Tenant}=t}$

$\bowtie_{\text{Product}}$

System Schema

TENANTPRODUCT	
Tenant	Product
1	1
1	3
2	2
2	4
3	4

PRODUCT		
Product	Name	…
1	Camera 5I	
2	TV 6000	
3	Camera 5II	
4	Notebook X7	

> Globally Shared Data

Tenant 1

PRODUCT		
Product	Name	…
1	Camera 5I	
2	TV 6000	
3	Camera 5II	

Tenant 2

PRODUCT		
Product	Name	…
2	TV 6000	

Tenant 3

PRODUCT		
Product	Name	…
2	TV 6000	
4	Notebook X7	

Virtual Schemas

$\sigma_{\text{Tenant}=t \lor \text{Tenant}=G}$

System Schema

PRODUCT			
Tenant	Product	Name	…
1	1	Camera 5I	
G	2	TV 6000	
1	3	Camera 5II	
3	4	Notebook X7	

> Common Data

Tenant 1

PRODUCT		
Product	Name	…
1	Camera 5I	
2	TV 6000	
3	Camera 5II	
4	Notebook X7	

Tenant 2

PRODUCT		
Product	Name	…
1	Camera 5I	1
2	TV 6000	2
3	Camera 5II	3
4	Notebook X7	4

Tenant 3

PRODUCT		
Product	Name	…
1	Camera 5I	1
2	TV 6000	2
3	Camera 5II	3
4	Notebook X7	4

Virtual Schemas

System Schema

PRODUCT		
Product	Name	…
1	Camera 5I	
2	TV 6000	
3	Camera 5II	
4	Notebook X7	

Customization

> Why Customizing?

Business applications

- Extremely complex
- Incorporates user-specific business rules
- As-it-is solution often not possible
- Customization during installation

Customization

- Different tenants may have a different view on the content
- Customizing an application often includes schema adjustments on database level
- May also reach as far out as changing common data

Example: Web shop

- Retailer wants to extend the product table with product-specific attributes
- Wine retailers need other attributes than TV retailers or book retailers

> Extension Tables

Background

- Big part of customization can be carried out in modules, extensions, or add-ons
- Tenants can book extensions to adapt the service to their needs
- Provider is in full control of the offered extension
- Provider can model the system tables of the database accordingly

Idea

- Decomposing system tables into base table and extension table
- Base table encompasses the columns every tenant needs
- Columns of an extension are combined in an extension table
- Tuples in an extension table reference their corresponding tuple in the base table

Advantages

- Data resides in a well modeled system schema
- Database features such as constraints, indexes, or aggregation are directly usable
- Avoids NULL values in the extension columns

Disadvantages

- Joins required to answer queries impose overhead for the virtualization layer
- Allows customization only in relatively coarse-grained, predefined modules

> Extension Tables (2)

Tenant 1

PRODUCT		
Product	Name	Sensor
1	Camera 5I	12MP
2	Camera 5II	18MP

Tenant 2

PRODUCT		
Product	Name	Screen
1	TV 6000	42”

…

Virtual Schemas

System Schema

$\sigma_{Tenant=1}$

$\bowtie_{Product,Tenant}$

$\sigma_{Tenant=2}$

$\bowtie_{Product,Tenant}$

CAMERAEXTENSION		
Tenant	Product	Sensor
1	1	12MP
1	2	18MP

TVEXTENSION		
Tenant	Product	Screen
2	1	42”

PRODUCT		
Tenant	Product	Name
1	1	Camera 5I
2	1	TV 6000
1	2	Camera 5II
3	1	Notebook X7

> Universal Table

Used by salesforce force.com

Idea

- A generic system schema allows consolidating arbitrary virtual schemas
- Universal table is one possible generic system schema
- System schema contains a single table
- Fixed number of generic byte string or char string columns
- Each tuple in the universal table represents one tuple in a virtual table of a tenant
- Column mapping between virtual tables and universal table stored in system catalog

Advantages

- Provides the most flexibility for customization
- Tenants can adapt their virtual schema without affecting the system schema
- Query rewriting procedure is simple; no additional joins

Disadvantages

- Necessary cast operations introduce a considerable overhead
- Very wide rows with many NULL
- No direct constraints and no index support (have to be implemented on top)
- Approach can be changed to technically typed columns → more NULL values

> Universal Table (2)

Tenant 1

PRODUCT		
Product	Name	Sensor
1	Camera 5I	12MP
2	Camera 5II	18MP

Tenant 2

PRODUCT		
Product	Name	Screen
1	TV 6000	42”

…

Virtual Schemas

System Schema

$\rho_{\text{Col1:Name,Col2:Sensor}}$

$\sigma_{\text{Tenant=1,Table=1}}$

$\rho_{\text{Col1:Name,Col2:Screen}}$

$\sigma_{\text{Tenant=2,Table=1}}$

UNIVERSALTABLE				
Tenant	Table	Col1	Col2	…
1	1	Camera 5I	12MP	…
2	1	TV 6000	32”	…
1	1	Camera 5II	18MP	…
3	2	…		

> Pivot Table

Idea

- Another generic system schema, also known as vertical schema
- Relational version of a triple store
- Single system table with five columns
  - Single generically typed column contains the values
  - Other four columns reference the tenant, the virtual table, the virtual column, and the virtual tuple
- Contains a tuple for each value in all virtual tables

Advantages

- Provides the most flexibility for customization
- Tenants can adapt their virtual schema without affecting the system schema
- No NULL values

Disadvantages

- Overhead: Querying n virtual column requires n-1 self-joins on system table
- Necessary cast operations introduce a considerable overhead
- No direct constraints and no index support (have to be implemented on top)
- Approach can be changed to technically typed columns → NULL values

> Pivot Table (2)

Tenant 1

Product		
Product	Name	Sensor
1	Camera 5I	12MP
2	Camera 5II	18MP

Tenant 2

Product		
Product	Name	Screen
1	TV 6000	42”

…

Virtual Schemas

System Schema

$$\sigma_{Tenant=1, Table=1}$$

$$\sigma_{Tenant=2, Table=1}$$

…

$$\bowtie_{Tenant, Table, Row}$$

$$\rho_{Value:Product} \qquad \rho_{Value:Name} \qquad \rho_{Value:Sensor}$$

$$\sigma_{Col='Product'} \qquad \sigma_{Col='Name'} \qquad \sigma_{Col='Senosor'}$$

PivotTable				
Tenant	Table	Row	Col	Value
1	1	1	Product	1
1	1	1	Name	Camera 5I
1	1	1	Sensor	12MP
2	1	2	…	…

> Chunk Table

Idea

- Combination of universal table and pivot table; semi-generic system schema
- Some combinations of technical types, e.g., int–string pairs, reoccur in virtual tables
- Decompose tenant data into chunks of frequent technical type combinations
- One chunk table for each frequent combination
- Each virtual column is mapped to a defined chunk table and chunk number
- A tenant tuple can be spread across multiple chunk tables and multiple chunk in same chunk table

Advantages

- Considerably more flexibility than Extension Tables
- Less joins need than on Pivot Table
- No NULL values
- Technical typed columns avoid expensive casts and allow index support

Disadvantages

- No arbitrary customizations: virtual table has to fit on the available chunk tables
- For full flexibility requires strategy to handle cutoff chunks
- Constraints have to be implemented on top
- How to select chunks?

> Chunk Table (2)

Tenant 1

Product		
Product	Name	Sensor
1	Camera 5I	12MP
2	Camera 5II	18MP

Tenant 2

Product		
Product	Name	Screen
1	TV 6000	42”

…

Virtual Schemas

System Schema

$\sigma_{Tenant=1,Table=1}$

$\bowtie_{Tenant,Table,Chunk}$

$\rho_{Int:Product,Str:Name}$ $\rho_{Int:Sensor}$

$\sigma_{Chunk=1}$ $\sigma_{Chunk=2}$

$\sigma_{Tenant=2,Table=1}$

$\bowtie_{Tenant,Table,Chunk}$

$\rho_{Int:Screen}$ $\rho_{Int:Product,Str:Name}$

$\sigma_{Chunk=2}$ $\sigma_{Chunk=1}$

ChunkTable					
Tenant	Table	Row	Chunk	Int	Str
1	1	1	1	1	Camera 5I
1	1	1	2	12MP	%
2	1	1	1	1	TV 6000
2	1	1	2	42”	%

> Interpreted Column

Idea

- Easy solution to add columns to a well-modeled database schema
- Well-modeled part is mapped one-to-one
- Custom columns are serialized to text (values + reference to column definition)
- Possible serializations XML, JSON, CSV, etc.
- System tables contain additional text column to take serialized columns

Advantages

- Flexibility without sacrificing DBMS capabilities for the common part of schema
- Some DBMS have already XML support including query, update, constraints, indexing
- Lean virtualization layer

Disadvantages

- Overhead arise from the serialization/de-serialization
- Without DBMS support
  - Extra data movement between DBMS and virtualization layer
  - No support for constraints or indexing

> Interpreted Column (2)

Tenant 1

PRODUCT		
Product	Name	Sensor
1	Camera 5I	12MP
2	Camera 5II	18MP

Tenant 2

PRODUCT		
Product	Name	Screen
1	TV 6000	42”

…

Virtual Schemas

System Schema

$\sigma_{\text{Tenant}=1}$

$\pi_{*,\text{Sensor}=\text{EXTRACT}(\text{XML}, /\text{sensor::text}())}$

$\sigma_{\text{Tenant}=2}$

$\pi_{*,\text{Screen}=\text{EXTRACT}(\text{XML}, /\text{screen::text}())}$

PRODUCT			
Tenant	Product	Name	XML
1	1	Camera 5I	<sensor>12MP</sensor>
2	1	TV 6000	<screen>42”</screen>
1	2	Camera 5II	<sensor>18MP</sensor>
3	1	Notebook X7	…

> Interpreted Record

Idea

- Support for flexible structured tuples in DBMS
- Mark every value with an explicit column reference

Advantages

- No extra overhead in the virtualization layer
- Full support for constraints and indexing
- No NULL values

Disadvantages

- Storage overhead for column reference
- Gained flexibility limited to adding and removing columns
- No consolidation of complete individual tenant schemas

> Interpreted Record (2)

Tenant 1

Product		
Product	Name	Sensor
1	Camera 5I	12MP
2	Camera 5II	18MP

Tenant 2

Product		
Product	Name	Screen
1	TV 6000	42”

…

Virtual Schemas

System Schema

$\sigma_{Tenant=1}$

$\sigma_{Tenant=2}$

Product			
Tenant	Product		
1	1	Name:Camera 5I	Sensor:12MP
2	1	Name:TV 6000	Screen:42”
1	2	Name:Camera 5II	Sensor:18MP
3	1	Name:Notebook X7	…

> Polymorphic Table

Idea

- Combines sharing and customization of **metadata and data** in a single concept
- Customization is similar to object inheritance (or specialization)
- Custom tables inherit structure and data of the tables they customizes
- Multiple customizations of the same base table form an inheritance hierarchy
- Polymorphic table represents complete inheritance hierarchy
- System database is the collection of all polymorphic tables

Advantages

- Covers multiple sharing patterns in one strike
- Consolidates metadata and data as much as possible
- Allows comprehensive customization of shared data
- Data resides in a well modeled system schema
- No NULL values
- Constraints, indexes, or aggregation operators directly usable

Disadvantages

- Processing overhead depends on the degree of customization
- The deeper the inheritance tree of customizations, the more joins required

> Polymorphic Table (2)

Tenant 1

PRODUCT		
Product	Name	Sensor
1	Camera 5I	12MP
2	Camera 5II	18MP

Tenant 2

PRODUCT		
Product	Name	Screen
1	X-Tablet	11”
4	TV 6000	42”

…

Virtual Schemas

System Schema

$\bowtie_{Product}$

∪

TENANT1	
Product	Sensor
2	12MP
3	18MP

PRODUCTBASETENANT1	
Product	Name
2 (modified)	Camera 5I
3	Camera 5II

PRODUCTBASE	
Product	Name
1	X-Tablet
2	DSLR 5I

∪

$\bowtie_{Product}$

PRODUCTBASETENANT2	
Product	Name
4	TV 6000

TENANT2
Product
1
4

∪

SCREENEXTENSION	
Product	Screen
1	11”
2	3”

SCREENEXTENSIONTENANT2	
Product	Screen
4	42”

> Problems so far…

Problems in Implementing Multi-Tenancy

- Resource contention among tenants
  - Resource governing to ensure fairness is difficult to implement
  - But malicious and inadvertently bad requests must be shut down
- Common practice is simply to forego operations whose resource usage can't be bounded in advance SLAs
- Access control among tenants
  - May have to rely on the application or mapping algorithms rather than the database
- Moving data for an individual tenant
  - For archiving and load balancing
  - Tuple-at-a-time operations are slow and resource intensive

> Summary

- Different forms of scaling: up, out, in
- Economy of scale by re-using processes: Mutliple tenants on single database instance
- Problem: Isolation
- Separate tenants by mapping many logical schemata to single physical schema
- Many schema mappings techniques
- Problem: Query transformation