Make WossiDiA Efficient Again

Alf-Christian Schering

February 18, 2020

Analysis (1)

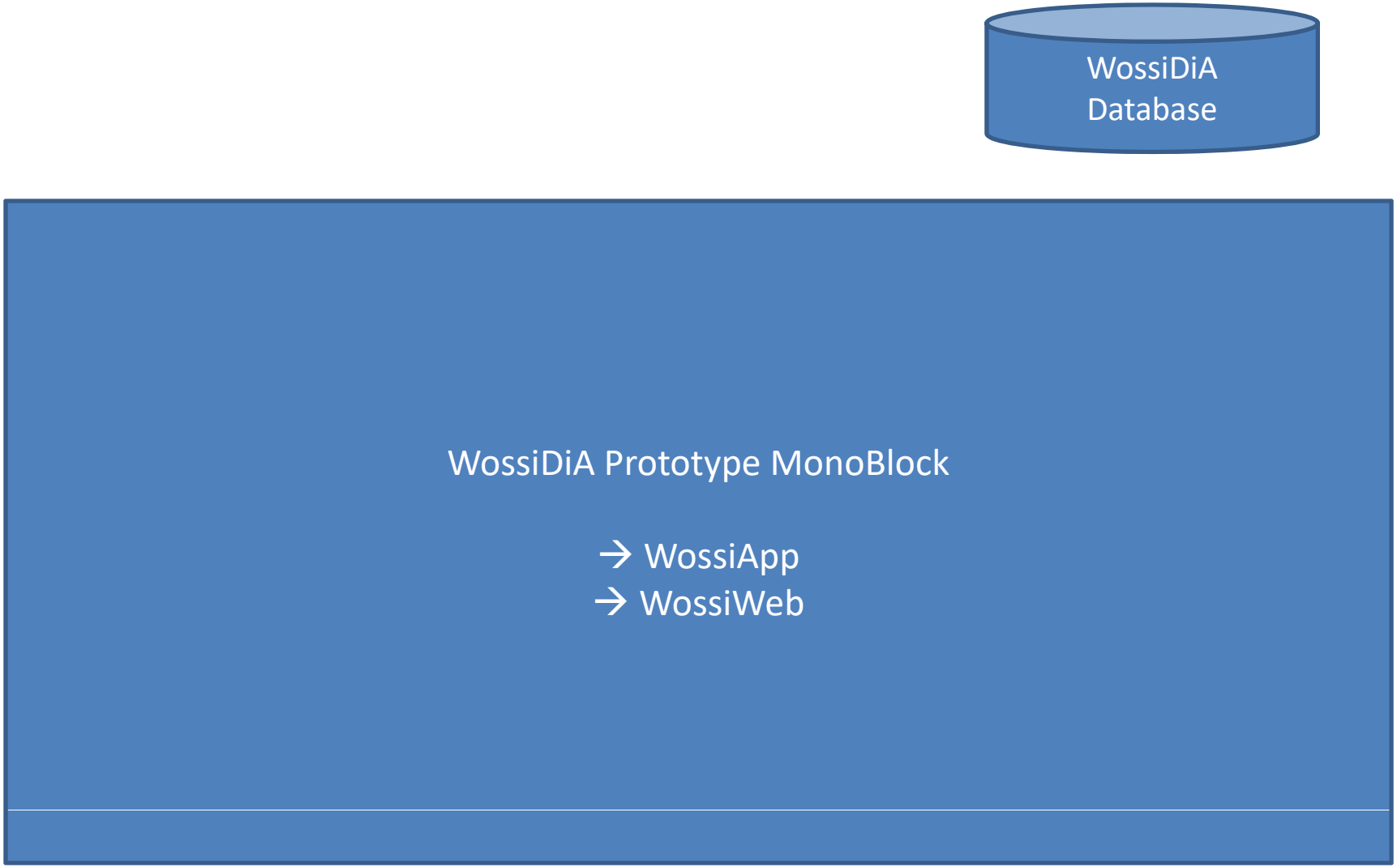
- Why this statement?
  - Resp. efficiency and usability of WossiDiA/PowerGraph user interfaces
    → there is still room for improvement
  - Increasing latencies in certain situations
  - New functionalities are not as fast as they should be!
  - Some well-known functionalities are not as fast as they used to be!
- How come?

  → Comparision of initial and current situation …

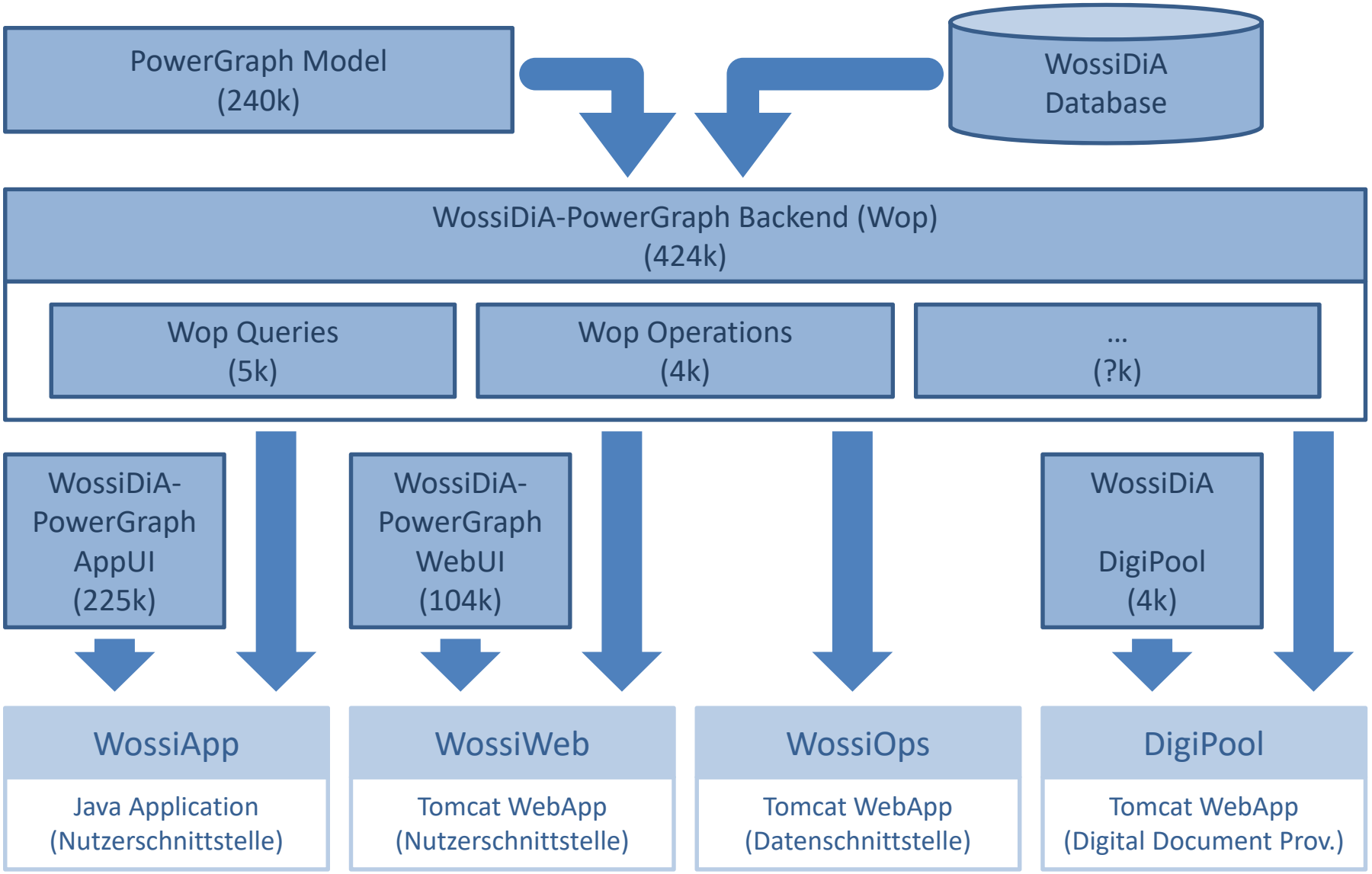
Analysis (2)

- Initial situation
  - Mono-block software
  - Highly specialized / many custom features
  - Few hypergraph data
- What happened?
  - Mono-block de-specialization / consolidation +++
  - Increased amount of hypergraph data
  - New and ISEBEL-specific challenges (new structures, use of paths)
- Current situation
  - Modularized packages
  - Much higher degree of separation between PowerGraph and WossiDiA application
- Consequences
  - Loss of very efficient custom data structures -> decreased efficiency
  - Increased data → decreased efficiency
  - → Users' pain levels increasing

System Architecture (until 2018)



System Architecture (since 2019)



Analysis (2)

- Initial situation
  - Mono-block software
  - Highly specialized / many custom features (eg. special topo attributes (n0, nn))
  - Few hypergraph data
- What happened?
  - Mono-block de-specialization / consolidation +++ (modularization)
  - Increased amount of hypergraph data
  - New and ISEBEL-specific challenges (new structures, use of paths)
- Current situation
  - Modularized packages
  - Much higher degree of separation between PowerGraph and WossiDiA application
- Consequences
  - Loss of very efficient custom data structures -> decreased efficiency
  - Increased data → decreased efficiency
  - New Paths + many joins → less efficiency
  - → Users' pain levels increasing

Analysis (3)

- What to do about it?
  - „**Make WossiDiA Efficient Again**“

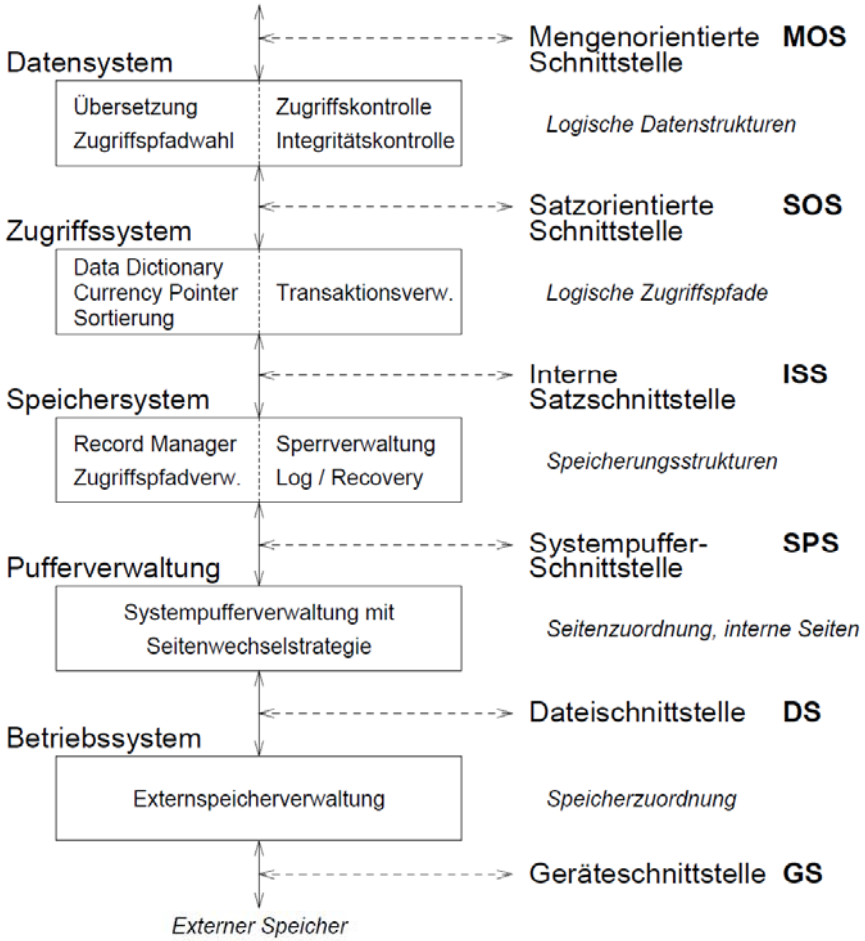
    by using classic database approaches to speed up „things“

  - (1) Establishing indexing facility in PowerGraph
    - Speed up access by magnitudes
  - (2) Implementing the client/server concept
    - Solitary relational backend access
    - Centralized access structures

Powergraph: Server/Client

- Effects
  - Solitary relational backend access
  - Centralized access structures
- Goal
  - Queries and updates shall be addressed to the PowerGraph server, only
  - No more „user-end graph operations“ accessing the relational PowerGraph backend
  - Implementation:
    Not REST, security-sensitive architecture required

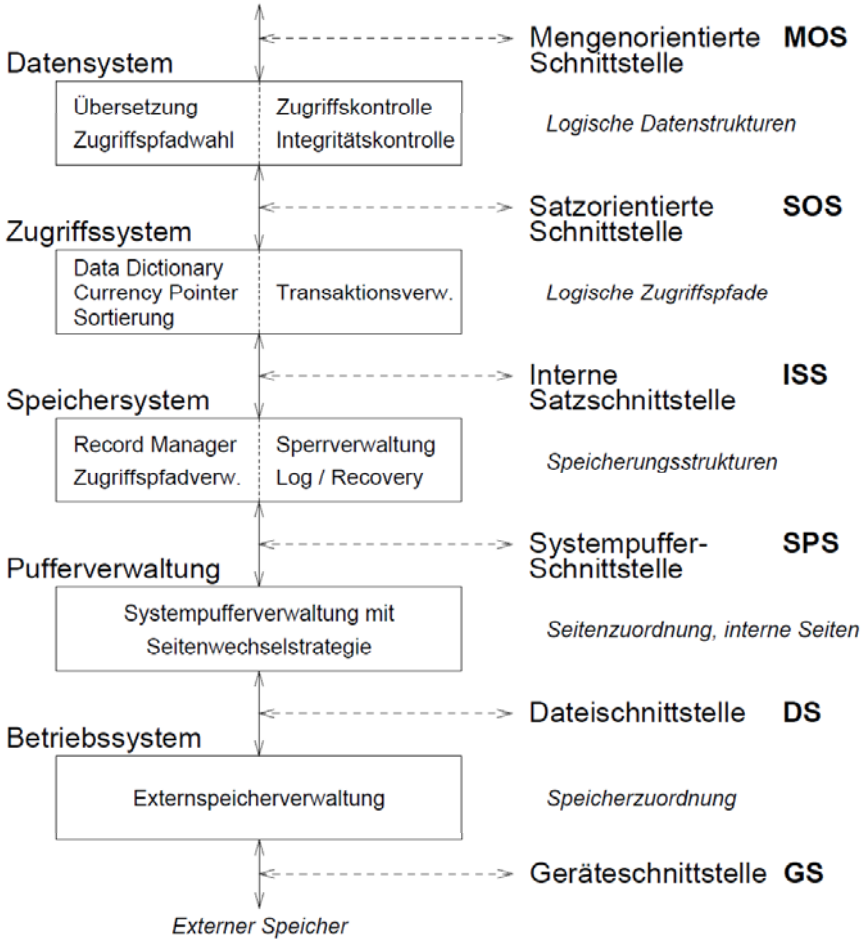
Five-Layer-Architecture



Source: DB2-Skript 2004

5LA: For WossiDiA-PowerGraph

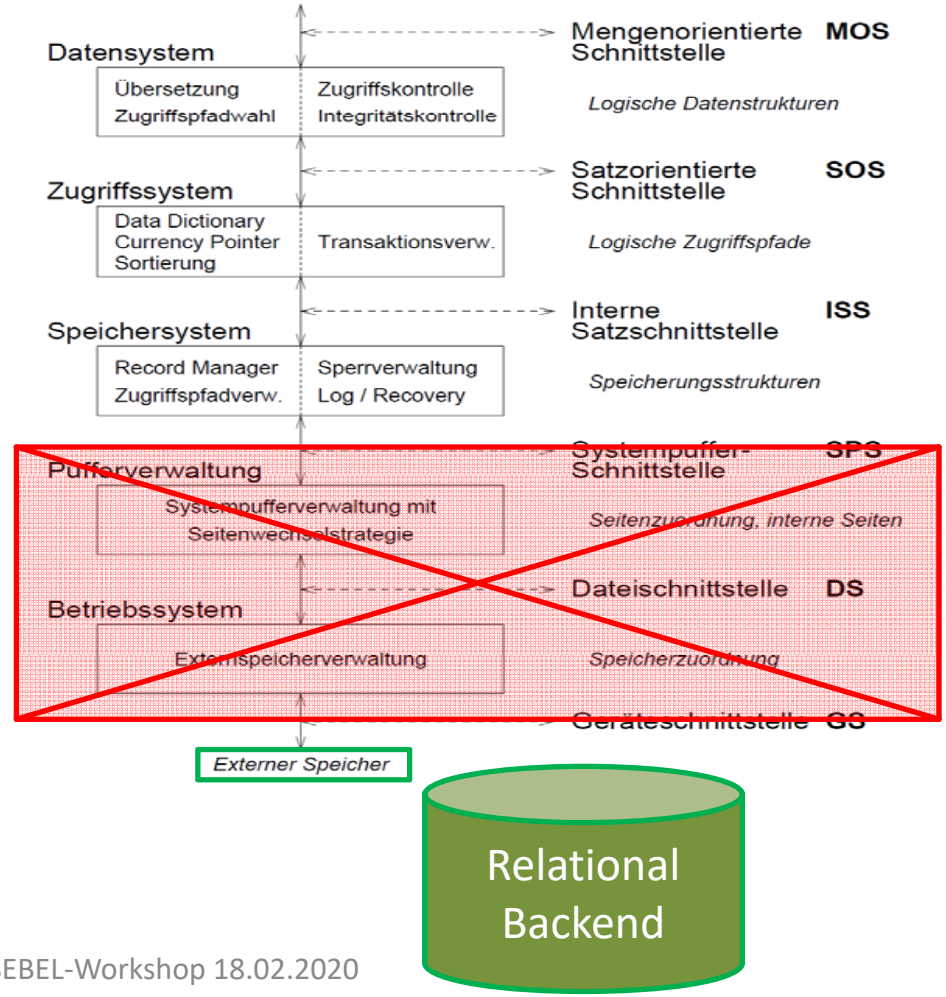
WOP applications



Source: DB2-Skript 2004

5LA: „Flimsy Analogy“

WOP applications

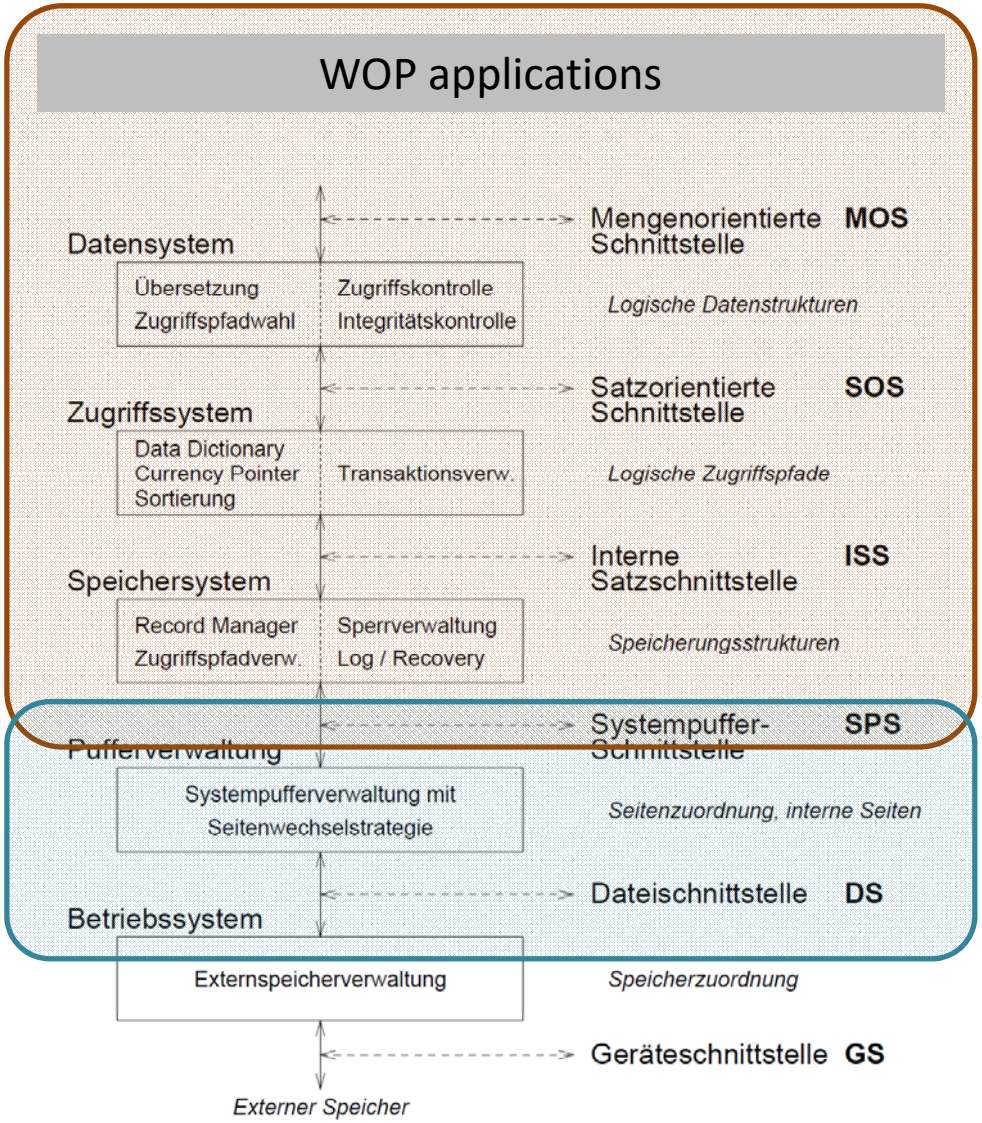


Source: DB2-Skript 2004

5LA: So far

WOP instances / every application

RDB backend



Source: DB2-Skript 2004

5LA: To be

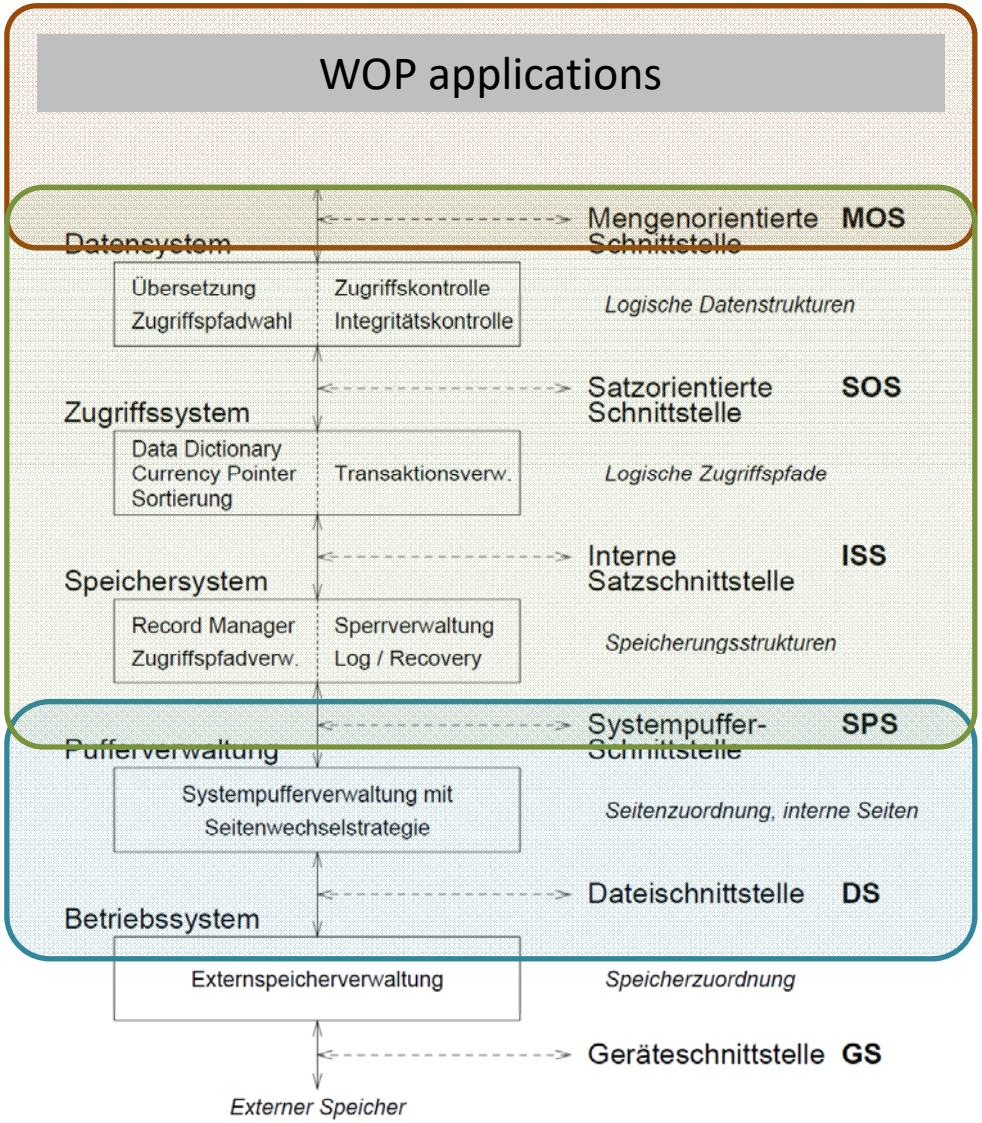
Clients: instances / every application

nx

WossiDiA PowerGraph Server

1x

RDB backend



Source: DB2-Skript 2004

Indexes for WossiDiA/PowerGraph

- Demand is due to the the differing aspects in WossiDiA/ISEBEL
- Currently: All index types are node-based
- Extension to other graph components on demand

	Index	Use case	Pain Index
N2N	Node to Node Path Index	ISEBEL stories, node templates, nodetype portals	4
TOP	Topology Index	WossiDiA topology	7
NAT	Node Abstract Text Index	Node string representations	8
LNK	Linking-Cache	General	G

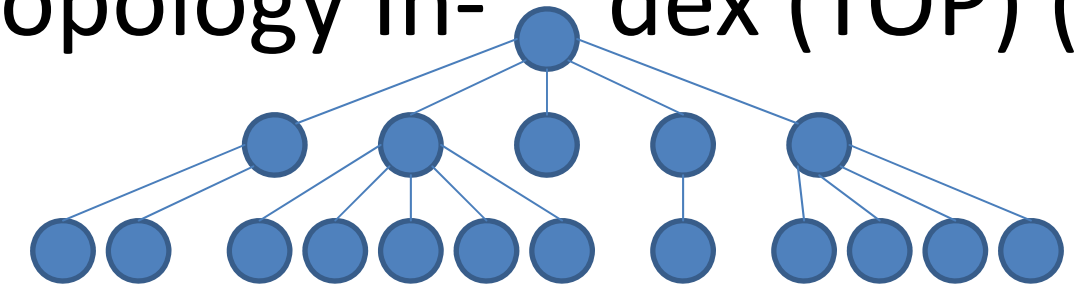
Node to Node Path Index (N2N)

- Use case
  - ISEBEL stories, node templates, nodetype portals
- Current implementation (about to be changed)
  - none
- Target implementation
  - Alternative 1: excl. nodes per path steps
    - N2N indexes start and end nodesof the path only
    - Nested index (n->n-m)
    - Symmetry: if applicable by creating the “opposite” index
  - Alternative 2: incl. nodes per path steps
    - Path index (n->n-*)
    - Symmetry: if applicable by creating the “opposite” index

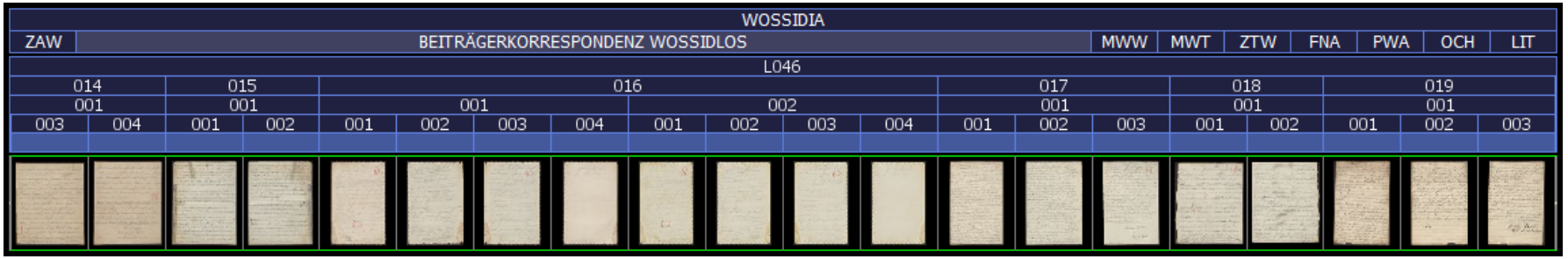
Topology Index (TOP) (1)

- Use case
  - WossiDiA topology
- Current implementation (about to be changed)
  - Special data structures for hierarchies (in WossiDiA)
    - Functionality as in compound index (n -> n-1 -> n)
    - Saves 4 joins per step
    - 100% (kompact in w.w_topdata)
  - Nested indexes (n -> n-m) and path indexes (n -> n-*) not present (synthesized by joint multi indexes / compound indexes)
- Target implementation
  - Hyperedge type for hierarchy
  - No longer topology storage
  - Topology support per path indexes, if applicable: compacted as Access Support Relation (ASR)

20200301 Supplemental: Topology In- dex (TOP) (2)



- Bonus problem: Tree node “coordinates” in linear 2D space
  - → Mapping from tree to rectangle in the plane
  - Problem: Unweighted trees, unbalanced
  - Paths not only down to bottom level
- So far:
  - Special attributes in topology relation (w.w_topdata)
- To be:
  → Must be respected in the index, too



Node Abstract Text Index (NAT)

- Use case
  - Node string representations
  - Must support full text seach
- Motivation
  - Massive fragmentation of data in hypergraph model
  - High demand of comprehensive, abstracted data, based on hypergraphs
  - → Integration of fragmented data
- Example: Person, Contribution, … definition: see →
- So far:
  - Special queries against the relational backend database (live)
  - Efficiency problem at runtime!!!
- To be:
  - Index: Node → Abstract Text
  - Text is full-text-indexed


```
{ "proj" : "signature || ' ' || coalesce(
coalesce(name, '')
|| CASE WHEN name IS NULL OR (gender IS NULL AND title IS NULL AND firstname IS NULL) THEN '' ELSE ',' END
|| CASE WHEN gender='m' THEN ' Herr' WHEN gender='f' THEN ' Frau' WHEN gender='l' THEN ' Fräulein' ELSE '' END
|| coalesce(' ' || title, '')
|| coalesce(' ' || firstname, '')
|| coalesce(' (' || oname || coalesce('/' || oregion, '') || ')', '')
|| coalesce(' (' || rname || ')', '')
|| CASE WHEN RoleCntr OR RoleSclr OR RoleNarr THEN (' ('
|| CASE WHEN RoleCntr THEN 'B' ELSE '_' END
|| CASE WHEN RoleSclr THEN 'G' ELSE '_' END
|| CASE WHEN RoleNarr THEN 'E' ELSE '_' END
|| ')') ELSE '' END
, 'Beiträger ' || sig0) || ' - Beitrag ' || sig1",
"name": "^<1:11:0>am_person.name",
"firstname": "^<1:11:0>am_person.firstname",
"gender": "^<1:11:0>am_person.gender",
"title": "^<1:11:0>am_person.title",
"RoleCntr": "^<1:11:0>am_person.RoleCntr",
"RoleSclr": "^<1:11:0>am_person.RoleSclr",
"RoleNarr": "^<1:11:0>am_person.RoleNarr",
"oname":"^<1:11:0>am_person<>am_place.name",
"oregion":"^<1:11:0>am_person<>am_place.region",
"rname":"^<1:11:0>am_person<>am_personrole.name",
"sig0": "^.sig0" }
```

Linking Cache (LNK)

- Use case
  - General
- Concept
  - Incidence matrix for nodes
  - Analogy: Compound index (n->n-1 / n-1->n)
  - Quickest possible access by full primary memory presence
- Alternative implementations (to be decided)
  - General incidence (any edge type)
  - Hyperedge-specific incidence (specific edge type)
  - If applicable: combined solution
- Benefits
  - Saves 1 to 4 joins
- Implementation
  - As soon as the client server architecture is alive

Indexes: Relational Storage (1) (N2N, TOP, NAT)

- Index specification (catalog) — w.i_spec
  - id — int — NN PK — Index ID
  - type — text — NN UN — Index type
  - name — text — NN UN — Index name
  - ntype — int — NN UN FK — Node type
  - spec — json — NN — Index definition

- Index data — w.i_data_*
  - index — int — NN PK — Index ID
  - node — int — NN PK — Node indexed
  - Data — int[] * — NN — Index data record

- … Alternative:
  - Array data type or JSON

  → Approach best supported by RDBS indexes will be picked

Indexes: Relational Storage (2) (N2N, TOP, NAT)

- Index data dependencies w.i_deps
  - index int NN PK Index ID
  - node int NN PK Node indexed
  - dep_node int * Node dependency
  - dep_edge int * Edge dependency
  - dep_link int * Edge link dependency
  - dep_attr int * Node/edge/~link attribute dependency

- Dirty graph components for pending index updates w.i_dirt
  - bad_node int * Dirty node
  - bad_edge int * Dirty edge
  - bad_link int * Dirty edge link
  - bad_attr int * Dirty node/edge/~link attribute
  - action text NN Modification action: added, changed, removed

* … no referential integrity here (to support remove operation)

20200303 Supplemental: JOIN-Matrix

DEPS				DIRT			
Node	Edge	Link	Attr	Node	Edge	Link	Attr
●			N	M	—	—	—
●			●	M	—	—	M
	●	N	N	—	M	—	—
	●	N	●	—	M	0	M
	●	●	N	—	M	M	—
	●	●	●	—	M	M	M

- Horizontal: Match criteria
- Vertical: OR

●	= NOT NULL
M	= Match
N/0	= NULL
—	= whatever

Indexes: Operations

- Global re-indexing
  - Manual
  - or by CREATE INDEX
- Continuous re-indexing (path- and graph component-dependent)
  - PGDB monitoring-triggered
- Query/Access
  - API (legacy), REST (legacy), Client!
- Queries:
  - Trigger re-indexing of affected nodes and those which's dependencies are affected by dirty graph components and have not yet been re-indexed

- EOF