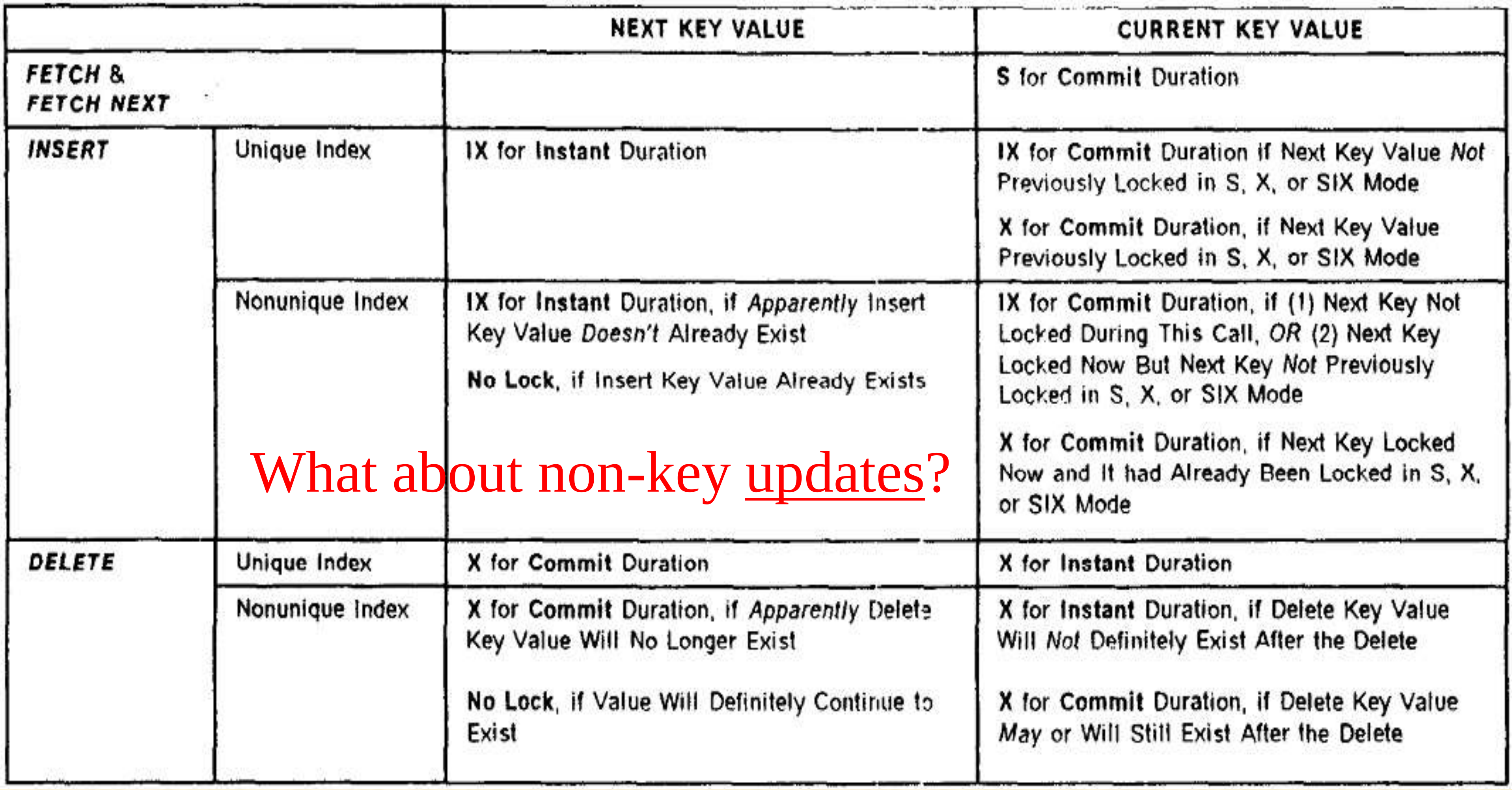
Orthogonal key-value locking

Goetz Graefe, Hideaki Kimura

Hewlett-Packard Laboratories

Palo Alto, Cal. – Madison, Wis.

ARIES/KVL



		NEXT KEY VALUE	CURRENT KEY VALUE
FETCH & FETCH NEXT			S for Commit Duration
INSERT	Unique Index	IX for Instant Duration	IX for Commit Duration if Next Key Value *Not* Previously Locked in S, X, or SIX Mode X for Commit Duration, if Next Key Value Previously Locked in S, X, or SIX Mode
	Nonunique Index	IX for Instant Duration, if *Apparently* Insert Key Value *Doesn't* Already Exist No Lock, if Insert Key Value Already Exists	IX for Commit Duration, if (1) Next Key Not Locked During This Call, *OR* (2) Next Key Locked Now But Next Key *Not* Previously Locked in S, X, or SIX Mode X for Commit Duration, if Next Key Locked Now and It had Already Been Locked in S, X, or SIX Mode
DELETE	Unique Index	X for Commit Duration	X for Instant Duration
	Nonunique Index	X for Commit Duration, if *Apparently* Delete Key Value Will No Longer Exist No Lock, if Value Will Definitely Continue to Exist	X for Instant Duration, if Delete Key Value Will *Not* Definitely Exist After the Delete X for Commit Duration, if Delete Key Value *May* or Will Still Exist After the Delete

What about non-key updates?

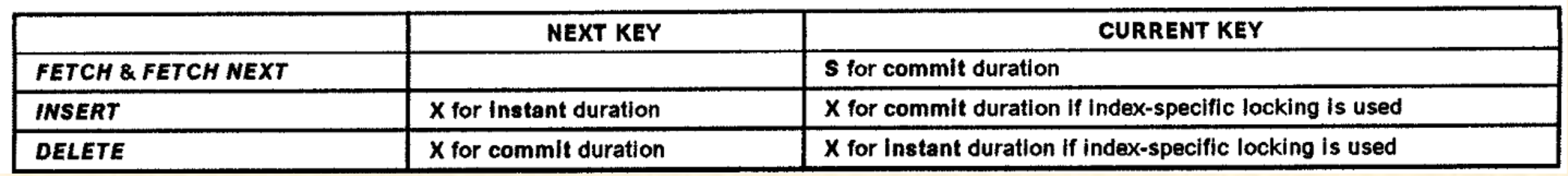
ARIES/IM

“Data-only locking”

- Logical row
- “Index-specific locking” in primary data structure

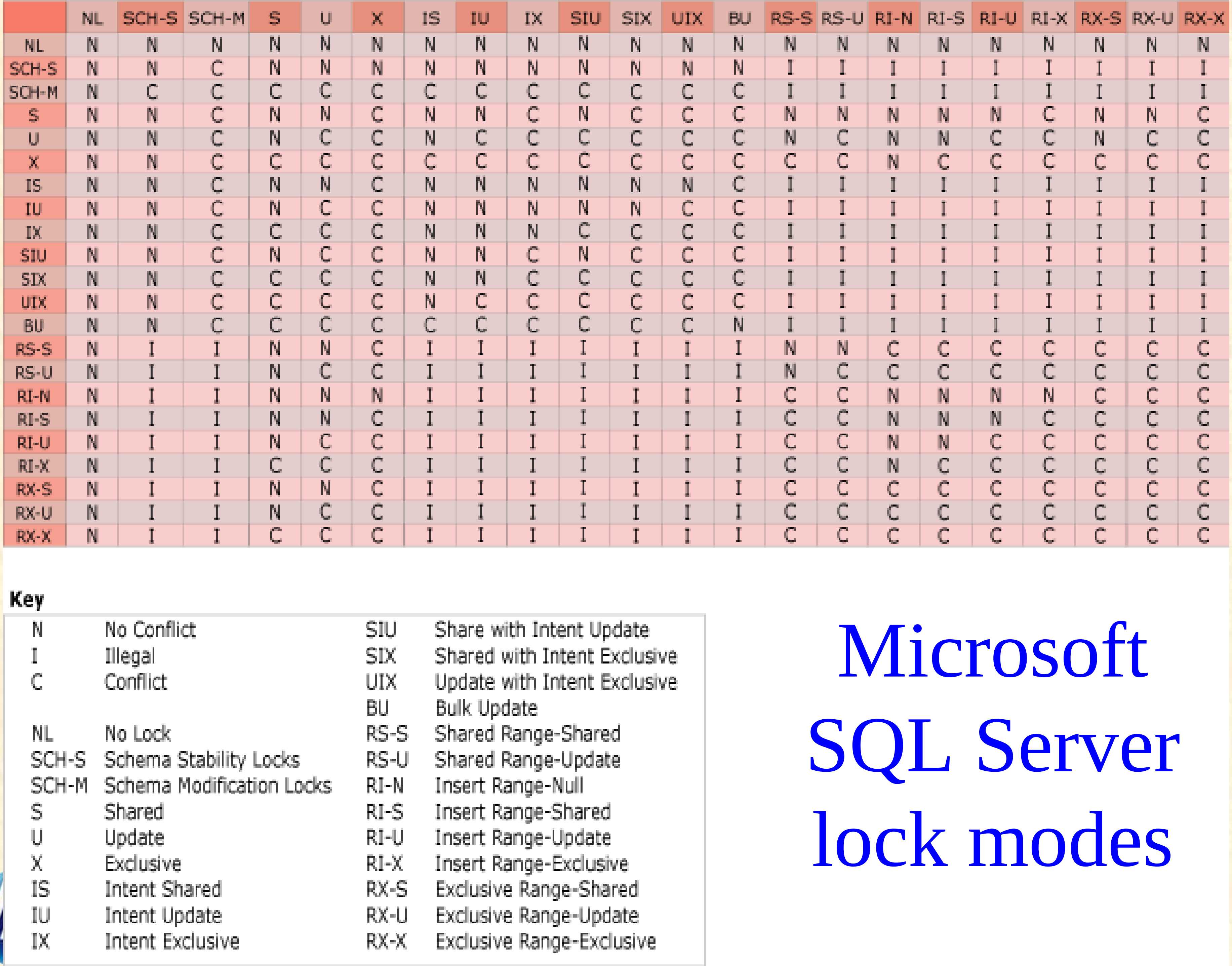
“Index-specific locking”

- Index entry + gap to next (lower) index entry



	NEXT KEY	CURRENT KEY
FETCH & FETCH NEXT		S for commit duration
INSERT	X for instant duration	X for commit duration if index-specific locking is used
DELETE	X for commit duration	X for instant duration if index-specific locking is used

What about non-key updates?



	NL	SCH-S	SCH-M	S	U	X	IS	IU	IX	SIU	SIX	UIX	BU	RS-S	RS-U	RI-N	RI-S	RI-U	RI-X	RX-S	RX-U	RX-X
NL	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N
SCH-S	N	N	C	N	N	N	N	N	N	N	N	N	N	I	I	I	I	I	I	I	I	I
SCH-M	N	C	C	C	C	C	C	C	C	C	C	C	C	I	I	I	I	I	I	I	I	I
S	N	N	C	N	N	C	N	N	C	N	C	C	C	N	N	N	N	N	C	N	N	C
U	N	N	C	N	C	C	N	C	C	C	C	C	C	N	C	N	N	C	C	N	C	C
X	N	N	C	C	C	C	C	C	C	C	C	C	C	C	C	N	C	C	C	C	C	C
IS	N	N	C	N	N	C	N	N	N	N	N	N	C	I	I	I	I	I	I	I	I	I
IU	N	N	C	N	C	C	N	N	N	N	N	C	C	I	I	I	I	I	I	I	I	I
IX	N	N	C	C	C	C	N	N	N	C	C	C	C	I	I	I	I	I	I	I	I	I
SIU	N	N	C	N	C	C	N	N	C	N	C	C	C	I	I	I	I	I	I	I	I	I
SIX	N	N	C	C	C	C	N	N	C	C	C	C	C	I	I	I	I	I	I	I	I	I
UIX	N	N	C	C	C	C	N	C	C	C	C	C	C	I	I	I	I	I	I	I	I	I
BU	N	N	C	C	C	C	C	C	C	C	C	C	N	I	I	I	I	I	I	I	I	I
RS-S	N	I	I	N	N	C	I	I	I	I	I	I	I	N	N	C	C	C	C	C	C	C
RS-U	N	I	I	N	C	C	I	I	I	I	I	I	I	N	C	C	C	C	C	C	C	C
RI-N	N	I	I	N	N	N	I	I	I	I	I	I	I	C	C	N	N	N	N	C	C	C
RI-S	N	I	I	N	N	C	I	I	I	I	I	I	I	C	C	N	N	N	C	C	C	C
RI-U	N	I	I	N	C	C	I	I	I	I	I	I	I	C	C	N	N	C	C	C	C	C
RI-X	N	I	I	C	C	C	I	I	I	I	I	I	I	C	C	N	C	C	C	C	C	C
RX-S	N	I	I	N	N	C	I	I	I	I	I	I	I	C	C	C	C	C	C	C	C	C
RX-U	N	I	I	N	C	C	I	I	I	I	I	I	I	C	C	C	C	C	C	C	C	C
RX-X	N	I	I	C	C	C	I	I	I	I	I	I	I	C	C	C	C	C	C	C	C	C

Key

N	No Conflict	SIU	Share with Intent Update
I	Illegal	SIX	Shared with Intent Exclusive
C	Conflict	UIX	Update with Intent Exclusive
		BU	Bulk Update
NL	No Lock	RS-S	Shared Range-Shared
SCH-S	Schema Stability Locks	RS-U	Shared Range-Update
SCH-M	Schema Modification Locks	RI-N	Insert Range-Null
S	Shared	RI-S	Insert Range-Shared
U	Update	RI-U	Insert Range-Update
X	Exclusive	RI-X	Insert Range-Exclusive
IS	Intent Shared	RX-S	Exclusive Range-Shared
IU	Intent Update	RX-U	Exclusive Range-Update
IX	Intent Exclusive	RX-X	Exclusive Range-Exclusive

Microsoft SQL Server lock modes

Orthogonal key-range locking

- Combine IS+S+Ø into SØ (“key shared, gap free”)

  Reduce lock manager invocations by factor 2-3

- Strict application of standard techniques

  No new semantics

  Automatic derivation

Gap

Key

	_Ø	_S	_X
Ø_	Ø	ØS	ØX
S_	SØ	S	SX
X_	XØ	XS	X

	S	X	SØ	ØS	XØ	ØX	SX	XS
S	ok		ok	ok				
X								
SØ	ok		ok	ok		ok	ok	
ØS	ok		ok	ok	ok			ok
XØ				ok		ok		
ØX			ok		ok			
SX			ok					
XS				ok				

Prior work leaves problems to solve:

- ARIES/KVL is complex and locks entire lists
  - Unable to lock individual entries: reduced concurrency
  - Poor support for phantom protection
- ARIES/IM locks much more than needed
  - Keys and gaps in multiple indexes
  - Very poor support for phantom protection
- SQL Server & Orthogonal KRL lock each entry
  - Many lock manager calls: late failure, lock escalation
  - Poor precision for equality queries

Goals to achieve

- Lock a distinct key value, i.e., an entire list ☺

  In a single lock manager invocation

  Including actual and possible list entries

  ⇒ key-value locking

- Lock a key value or a gap or both ☺

  In a single lock manager invocation

  ⇒ orthogonal lock modes

- Lock individual instances within a list ☹

  At least enable *some* concurrency within a list ☺

New technique: partitioned lock lists

- A list of entries per key
- Hash partitioning function on list entries
- A lock mode per partition
- A lock mode for the gap

Examples (4 partitions)

- Delete
  key value "Joe", row id 9
  Lock ("Joe", hash(9) % 4)
  or "Joe" in ØXØØØ
- Select key value "Joe"
  Lock "Joe" in SSSSØ

Case studies: example table

Emp No	First Name	Zip Code	Phone
1	Mike	42062	4567
2	Gary	10032	1122
3	Joe	46045	9999
4	Larry	53704	5347
5	Joe	67882	5432
…			

- Table with unique and non-unique columns and indexes
- Primary index on primary key: EmpNo
- Unique secondary index: Phone
- Secondary indexes: ZipCode, FirstName

Absence in a non-unique index

First Name	Count	EmpNo values…
Gary	1	2
Joe	2	3, 5
Larry	1	4
Mike	1	1

Select… FN="Henry"
Phantom protection

- ARIES/KVL
  All Joe values + gap to Gary
- ARIES/IM
  Row 3 + all lower gaps
- SQL Server KRL
  Joe:3 + gap to Gary:2
- Orthogonal KRL
  Gap above Gary:2
- Orthogonal KVL
  Gap above Gary (below Joe)

Selection in a non-unique index

First Name	Count	EmpNo values…
Gary	1	2
Joe	2	3, 5
Larry	1	4
Mike	1	1

Select… FN="Joe"
Successful selection

- ARIES/KVL
  All Joe values + gap to Gary
- ARIES/IM
  Rows 3, 5, 4,
  + 3 gaps in each index
- SQL Server KRL
  Joe:3, Joe:5, Larry:4, + 3 gaps
- Orthogonal KRL
  Gap above Gary:2,
  Joe:3, Joe:5, with gaps
- Orthogonal KVL
  All partitions of Joe, no gaps

Range queries in a non-unique index

First Name	Count	EmpNo values…
Gary	1	2
Joe	2	3, 5
Larry	1	4
Mike	1	1

Select… FN between
“Joe” and “Larry”

- ARIES/KVL
  Joe, Larry, + 2 gaps
- ARIES/IM
  Rows 3, 5, 4, 1: 4 rows
  + 4 gaps in each index
- SQL Server KRL
  Joe:3, Joe:5, Larry:4, Mike:1:
  4 keys + gaps
- Orthogonal KRL
  Gary:2, Joe:3, Joe:5, Larry:4:
  3 keys + 4 gaps
- Orthogonal KVL
  Joe, Larry: 2 keys + 1 gap

Non-key updates

First Name	Emp No	Zip Code
Gary	2	10032
Joe	3	46054
Joe	5	67882
Larry	4	53704
Mike	1	42062

Update ZipCode=…
where EmpNo=3

- ARIES/KVL
  X on Joe including gap
- ARIES/IM
  X on row 3 and all gaps
- SQL Server KRL
  X on Joe:3 and gap
- Orthogonal KRL
  X on Joe:3, not on gap
- Orthogonal KVL
  X on (Joe, hash (3) % k)

Deletion in a non-unique index

First Name	Count	EmpNo values…
Gary	1	2
Joe	2	3, 5
Larry	1	4
Mike	1	1

Delete… EmpNo=3

- ARIES/KVL
  X on Joe, no ghost
- ARIES/IM (via ghost)
  X on row 3 and all gaps
- SQL Server KRL
  X on Joe:3 and gap
- Orthogonal KRL
  X on Joe:3, not on gap
- Orthogonal KVL
  X on (Joe, hash (3) % k)

Insertion of an additional instance

First Name	Count	EmpNo values…
Gary	1	2
Joe	2	3, 5
Larry	1	4
Mike	1	1

Insert… (6, “Joe”, …)

- ARIES/KVL
  IX on Joe + gap
- ARIES/IM
  Instant X on 4 + X on 6
- SQL Server KRL
  Instant X on Larry:4 +
  X on Joe:6
- Orthogonal KRL
  Test ØX on Joe:5,
  then XØ on ghost Joe:6
- Orthogonal KVL
  X on (Joe, hash (6) % k)

Insertion of a new key value

First Name	**Count**	**EmpNo values…**
Gary	1	2
Joe	2	3, 5
Larry	1	4
Mike	1	1

Insert… (7, “Henry”, …)

- ARIES/KVL: complex!
- ARIES/IM: many gaps!
- SQL Server KRL
  Instant X on Joe:3
  X on Henry:7
- Orthogonal KRL
  Check gap above Gary:2
  X on ghost Henry:7
- Orthogonal KVL
  Check gap above Gary
  X on (Henry, hash (7))

Performance

- TPC-C customer table

  Non-unique secondary index on (w_id, d_id, last, first, id)

  3,000 customers per warehouse & district

  3 customers per last name
- HP workstation

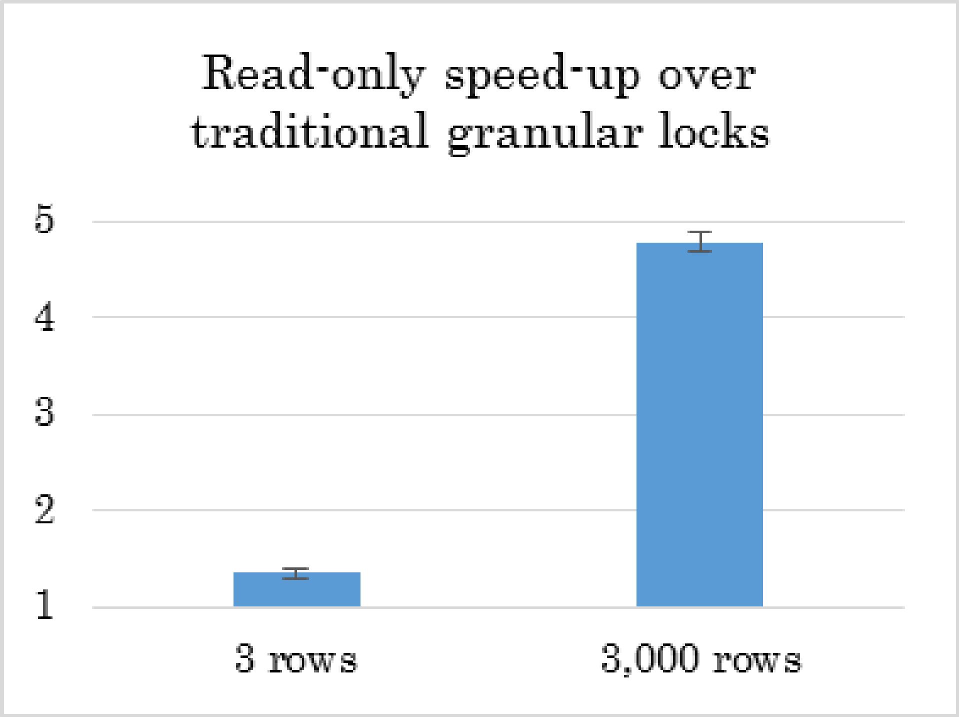
  HP Z820 Xeon

  2×8 cores, 3.4 GHz, 128 GB

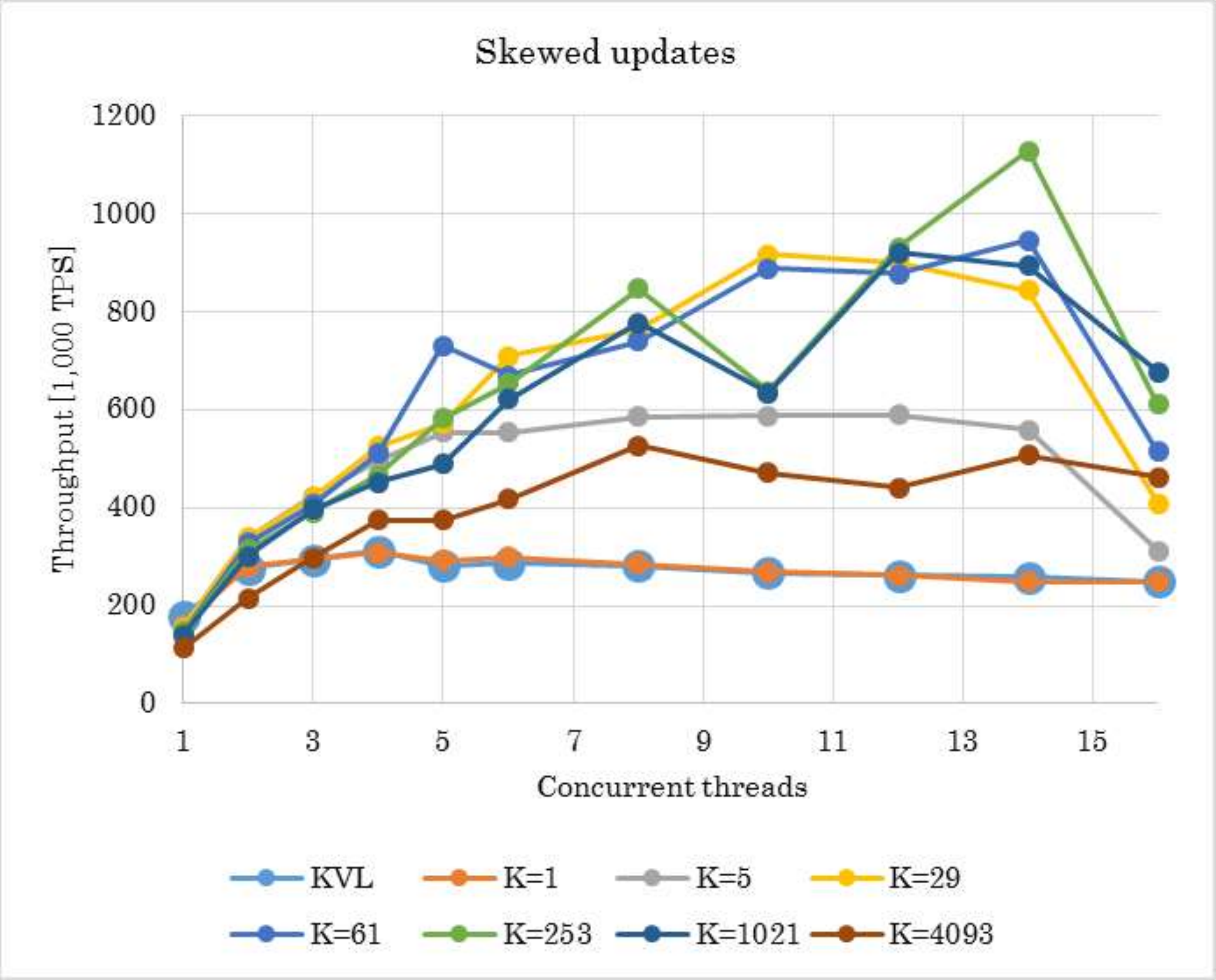
Shore-MT with many performance improvements

- Ghost records
- System transactions
- Foster b-trees
- Buffer pool with swizzled parent-to-child pointers
- Log with flush pipeline & consolidation array
- Read-after-write lock management

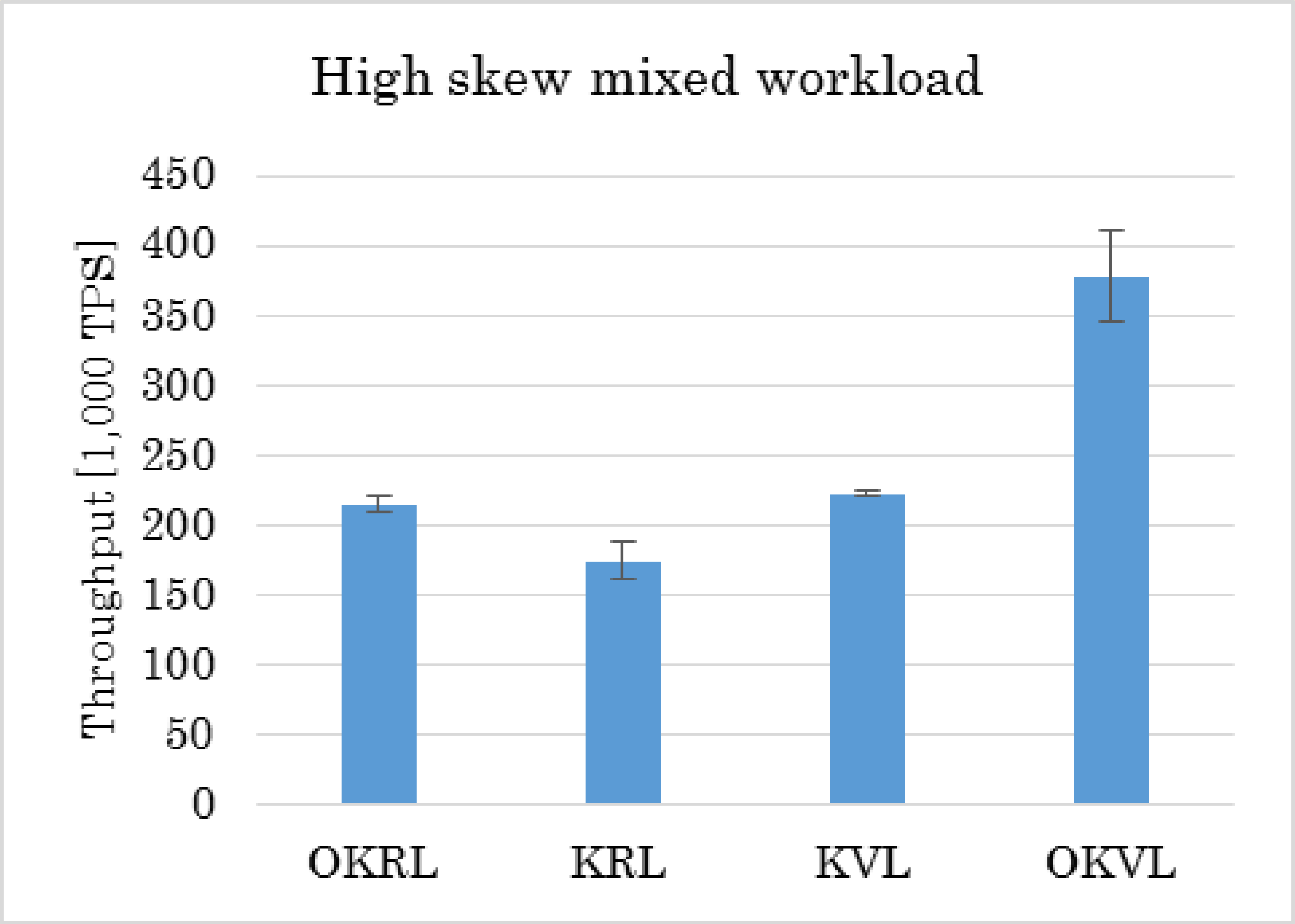
Read-only cursor, equality predicate



Updates only – lots of contention



Read-write workload with contention



Orthogonal key-value locking

Techniques

- Hybrid of traditional KVL and orthogonal KRL
- Fixed # of partitions in each list of bookmarks
- A single request can lock:

  a key value's entire list –
  all *possible* instances

  a single partition within a list

  a gap between distinct keys

  any combination, eg list + gap

Comparisons

- Fewest lock requests

  Better than ARIES/IM,
  SQL Server, orth KRL

- Precise locks for queries

  Better than ARIES/KVL, /IM,
  SQL Server, orth KRL

- High update concurrency

  Better than ARIES/KVL, /IM

- Fewest lock modes

  Better than SQL Server,
  orthogonal key-range locks

Why research granularity of locking?

Bad reputation of locking

- Too much overhead

  Is this actually true?

- Too little concurrency

  Poorly chosen lock modes

  Poorly chosen granules

  Excessive lock duration

- Weak isolation levels

  Dirty read, read committed

  Eventual consistency

Recommendation

- Strict serializability

  Easy application development

- Read-only transactions in snapshot isolation:
  commit point = start-of-tx

- All other transactions lock:
  commit point = end-of-tx

  Minimal lock duration

  Optimal lock modes

  Optimal granularity of locking

Design	Origin	Granularity	Comments
ARIES/KVL	IBM 1990	Distinct key value	All possible instances Incl gap to next lower “Instant duration locks”
ARIES/IM “data only locking”	IBM 1992	Logical row	Heap record + all index entries + gaps to next lower
ARIES/IM per index		Index entry	Incl gap to next lower
Key-range locking	DEC 1993	Index entry Range	First key-gap separation “Insertion” lock mode
Orthogonal key-range locking	Msft 2006	Index entry Gap	Cartesian product – simple derivation of locks & compatibility
Orthogonal key-value locking	HP 2013	Distinct key Partition Gap	All possible instances Hierarchy: key value + partitions
Orthogonal row locking	HP 2015	Logical row Index entry(ies) Gap or gaps	To be done…