Don’t Stop the World

2017.5.25

Takashi HORIKAWA

Who am I

Name

Takashi HORIKAWA, Ph. D.

Research interests

Performance evaluation of computer & communication systems, including

performance engineering of IT systems

with slightly shifting the focus of the research to CPU scalability

Papers

Non-volatile Memory (NVM) Logging, PGCon 2016

Latch-free data structures for DBMS: design, implementation, and evaluation, SIGMOD ‘13

An Unexpected Scalability Bottleneck in a DBMS: A Hidden Pitfall in Implementing Mutual Exclusion, PDCS ‘11

An approach for scalability-bottleneck solution: identification and elimination of scalability bottlenecks in a DBMS, ICPE '11

A method for analysis and solution of scalability bottleneck in DBMS, SoICT '10

Agenda

Introduction

Start with trend in computer architecture

Simple is best

CLogControlLock

Prepare in advance

Table extension

Achievements to date (Review)

Countermeasures for CPU scalability bottlenecks

Concluding remarks

Agenda

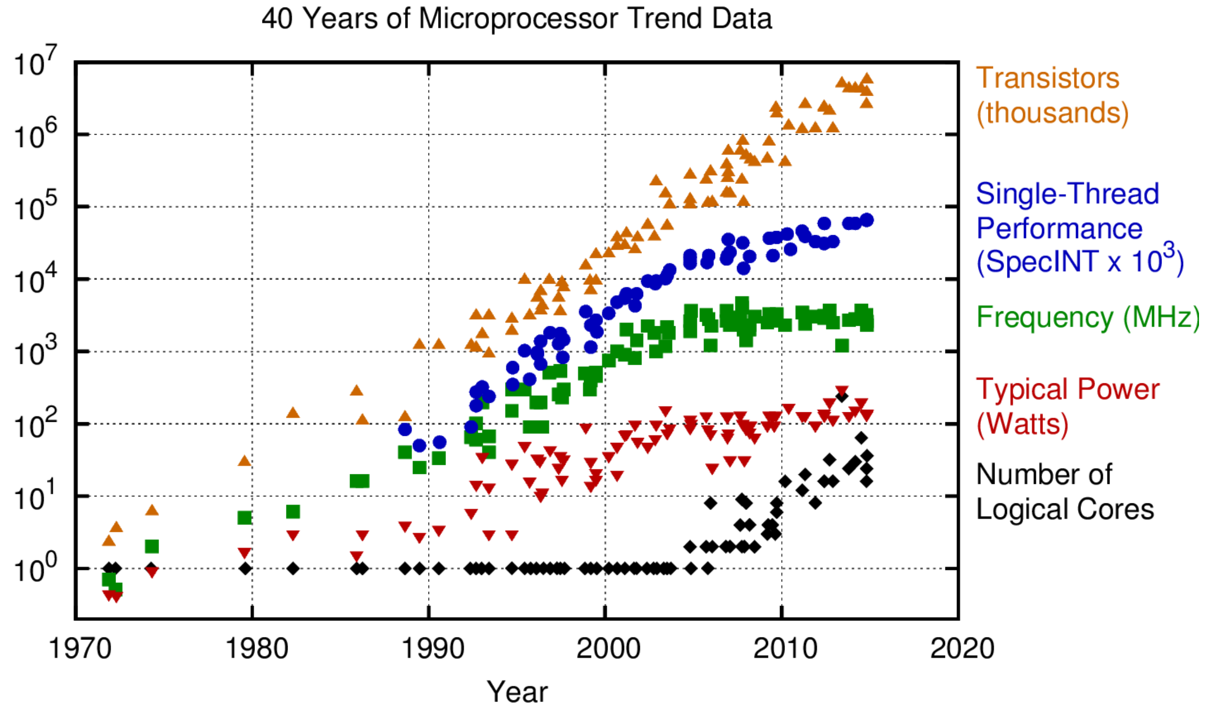
- Introduction
  - Start with trend in computer architecture
- Simple is best
  - CLogControlLock
- Prepare in advance
  - Table extension
- Achievements to date (Review)
  - Countermeasures for CPU scalability bottlenecks
- Concluding remarks

Trend in computer architecture

- Single core performance is saturated
- Core count is increasing

We have to benefit from lots of CPU cores.

Parallelism



Original data up to the year 2010 collected and plotted by M. Horowitz, F. Labonte, O. Shacham, K. Olukotun, L. Hammond, and C. Batten
New plot and data collected for 2010-2015 by K. Rupp

https://www.karlrupp.net/2015/06/40-years-of-microprocessor-trend-data/

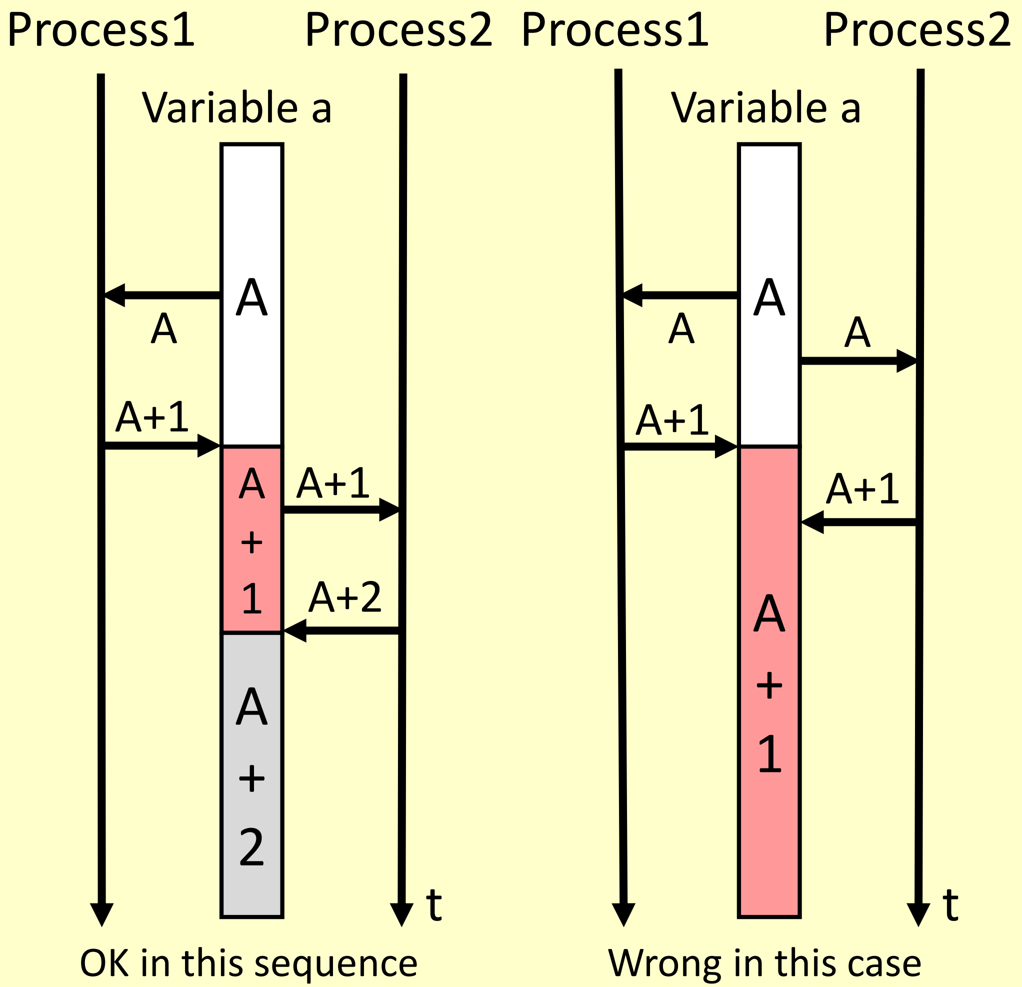
Utilization of Parallelism

- ‘Read’ and ‘write’ write have completely different aspects

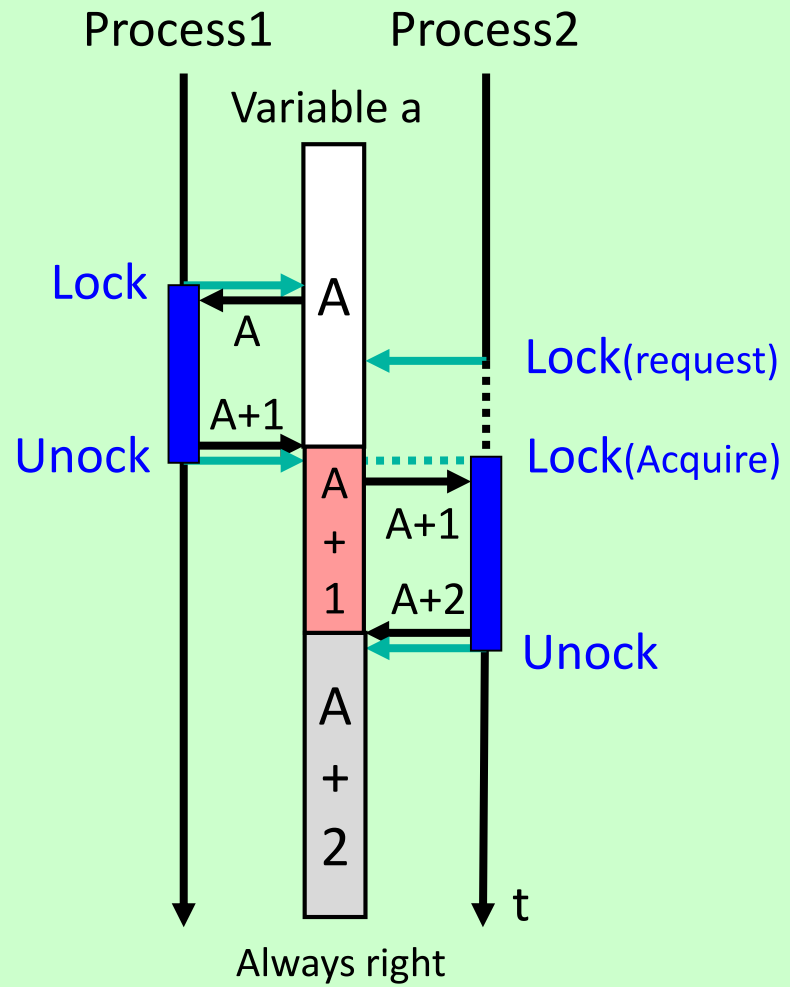
Trx. Type	ACID property	Granularity of parallelism	Point of interest
Write	Careful control required	Transaction	Tuning the critical sections
Read (only)	No need to worry about ACID	Operator in plan tree	Parallel query

Focus here

Why is a critical section necessary?



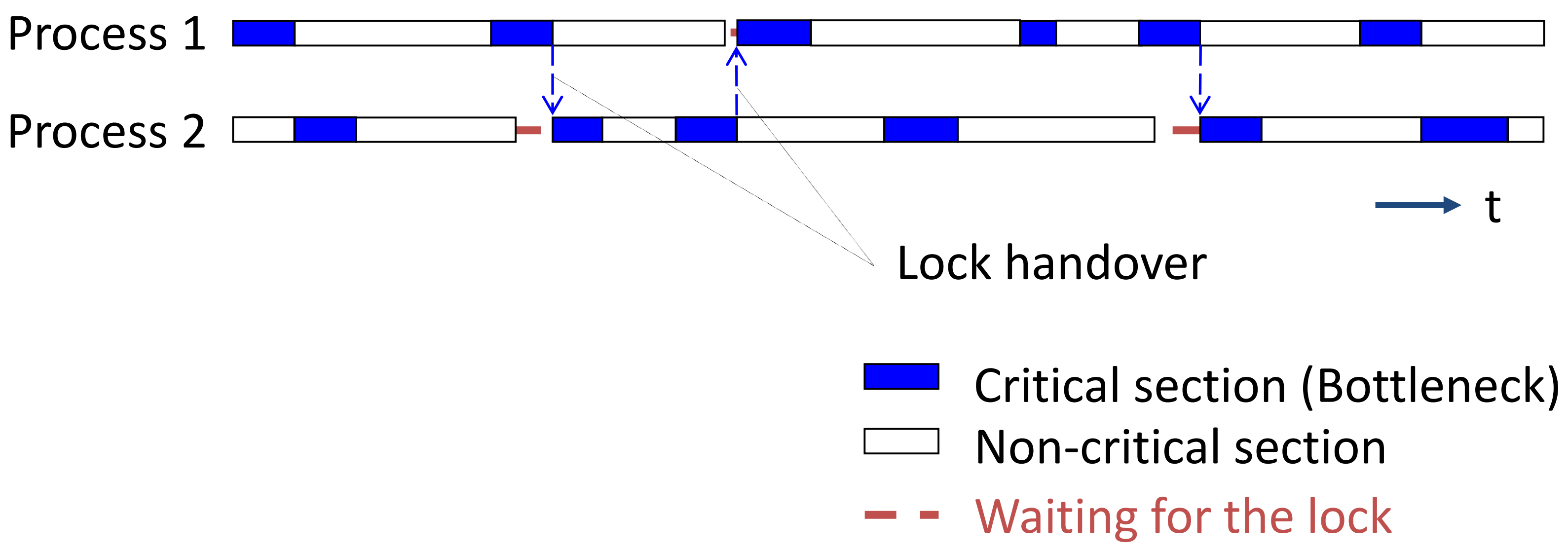
Without critical section



Using critical section

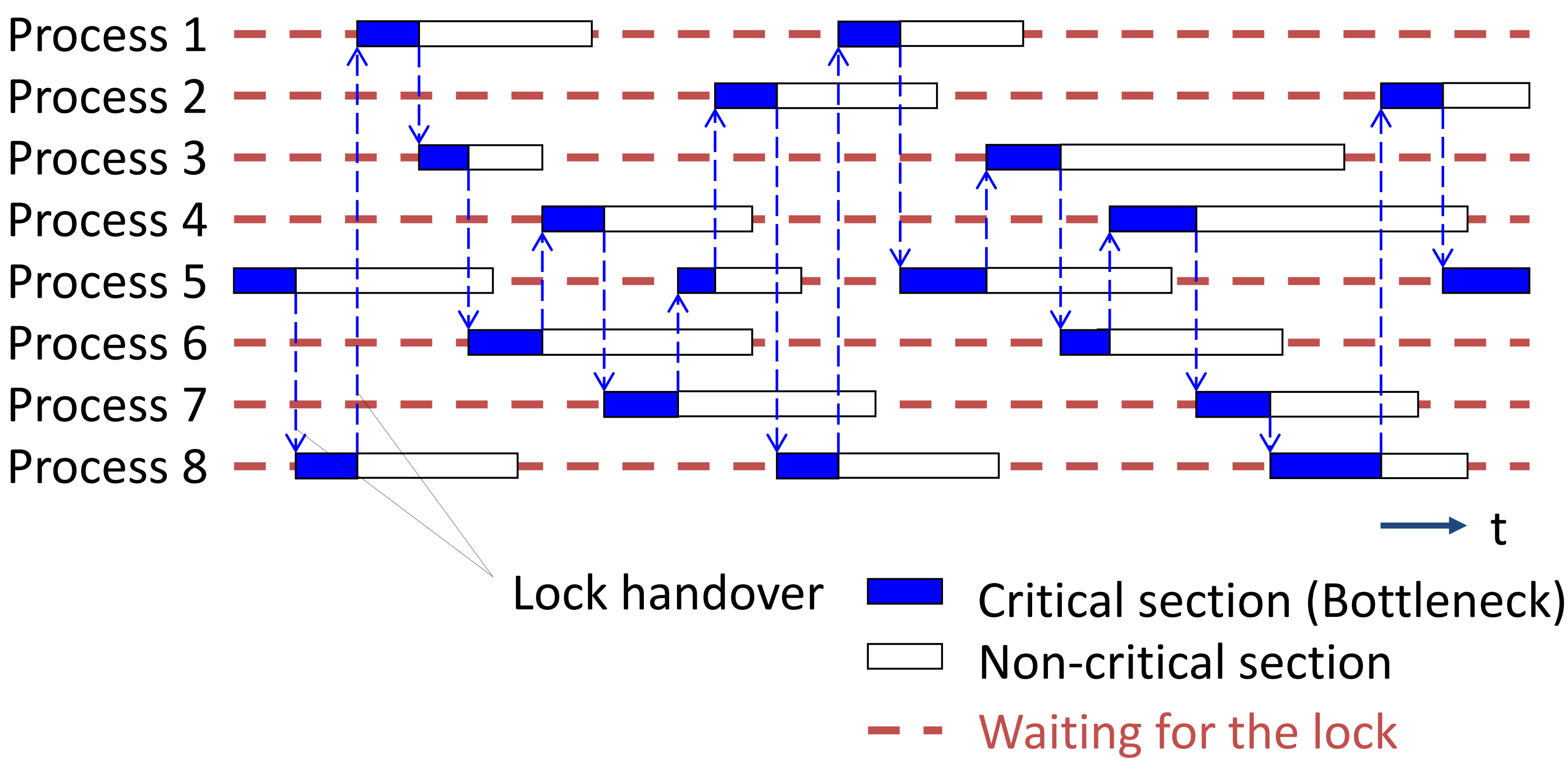
Critical section in small core systems

- Contention on the critical section is rare
  - Every process can work most of the time
  - Its Adverse effect is negligible



Behavior in many-core servers

- 'Stop the world operation' becomes prominent
  - Every process has a large lock wait time and can not do any useful work



Amdahl’s Law

- Amdahl's Law is a law governing the *speedup* of using parallel processors on a problem, versus using only one serial processor.

$$S = \frac{N}{(B*N) + (1-B)}$$

S : Speedup
N : Number of processors
B : % of algorithm that is *serial*

https://home.wlu.edu/~whaleyt/classes/parallel/topics/amdahl.html

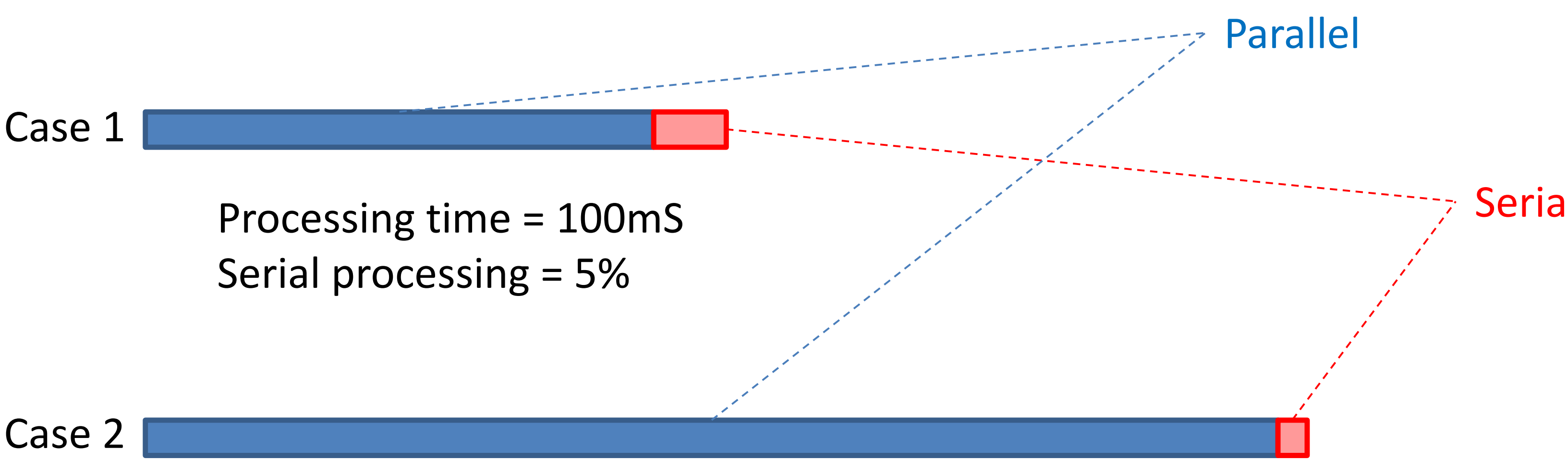
B = 0% -> S = N (Ideal)
B = 100% -> S = 1 (No gain)

⇒ Reduction of B is indispensable

Quiz

- Which is better?

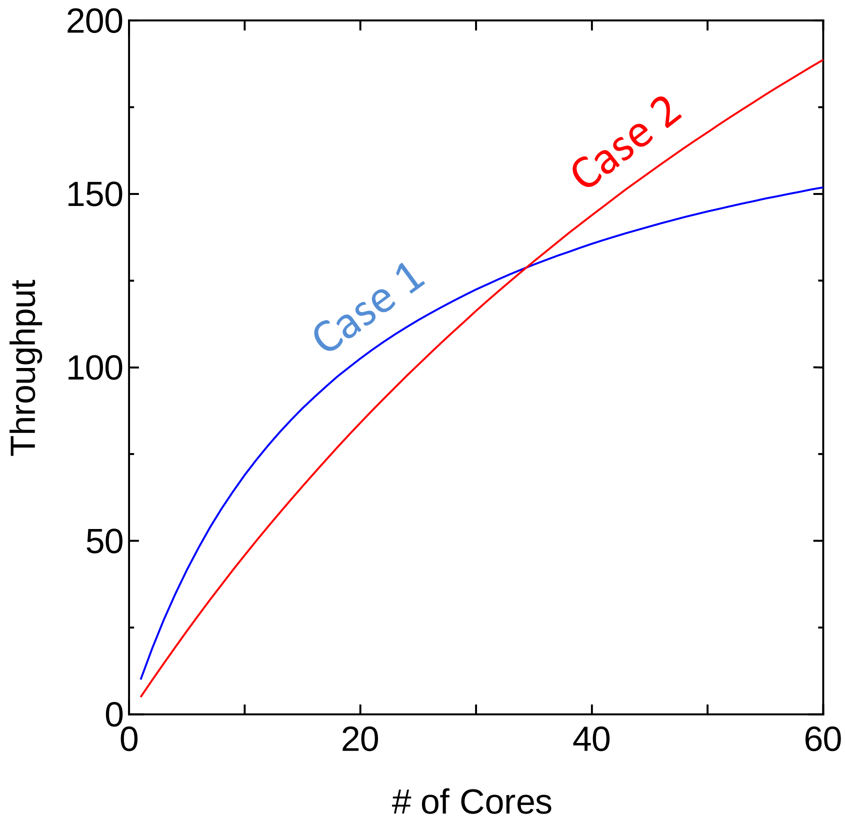
  i.e. Higher throughput/Shorter response time



Processing time = 100mS
Serial processing = 5%

Processing time = 200mS
Serial processing = 1%

There is no universal answer



The more the number of processors increase,
the more the effect of the serial execution larger.

Even if the amount of overall processing increases, it is better to reduce the serial execution part.

Degree of attention for LWLocks

Name of LWLock	Appearance count in www.postgresql.org
WALInsertLock	686
WALWriteLock	360
CLogControlLock	284
XidGenLock	168
ProcArrayLock	128
SerializableXactHashLock	82
ControlFileLock	80
SInvalReadLock	57
CheckpointLock	50
(WALBufMappingLock)	(22)

Main topic in this talk

As of 2017/5/5

Agenda

- Introduction
  - Start with trend in computer architecture
- Simple is best
  - CLogControlLock
- Prepare in advance
  - Table extension
- Achievements to date (Review)
  - Countermeasures for CPU scalability bottlenecks
- Concluding remarks

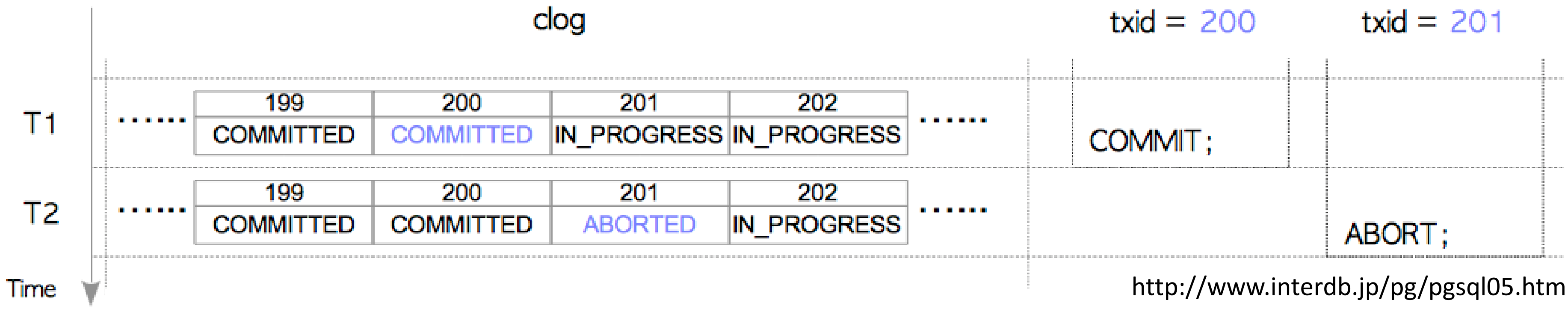
CLOG (Commit LOG)

- To implement MVCC, visibility of a tuple is determined using
  1. the t_xmin and t_xmax of the tuple,
  2. the **transaction status** (for t_xmin and/or t_xmax), and
  3. the obtained transaction snapshot.

Based on http://www.interdb.jp/pg/pgsql05.html

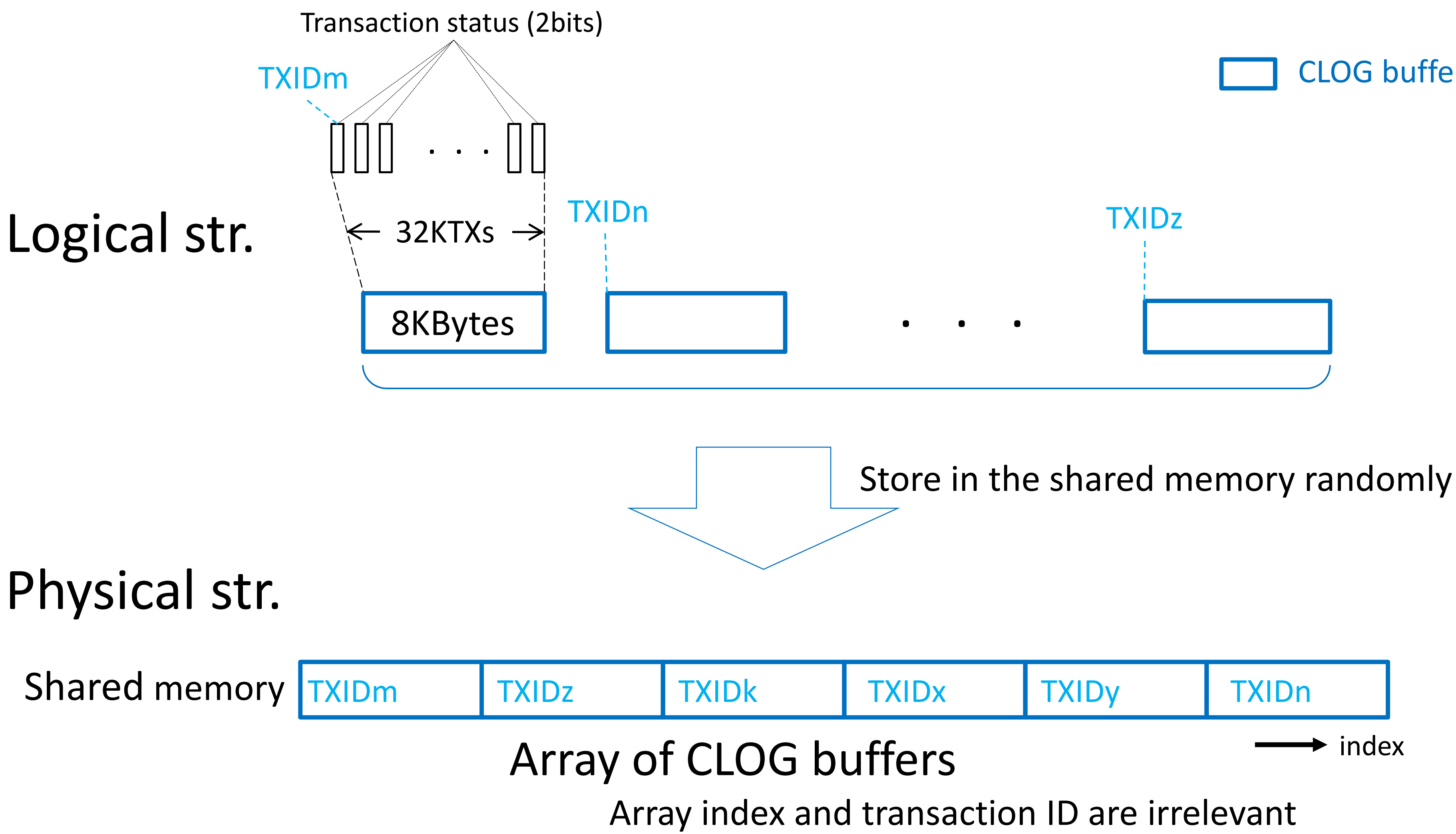
- Transaction status is maintained in **CLOG** which resides in $Data/pg_clog.
  – This table contains **two bits** of status information **for each transaction**; the possible states are in-progress, committed, or aborted.

https://www.enterprisedb.com/well-known-databases-use-different-approaches-mvcc



http://www.interdb.jp/pg/pgsql05.html

CLOG in shared memory



Points in CLOG buffer management

- Recently accessed CLOG Buffers are stored in the shared memory in random order
  - Trade off between hit ratio and search overhead
    - Buffer replacement based on LRU
    - Linear search
- History of the number of CLOG buffers

PG Version	- 9.1	9.2 – 9.5	9.6
Count	8	32	128

(Value when shared buffer is sufficiently large)

```
/*
 * Number of shared CLOG buffers.
 *
 * On larger multi-processor systems, it is possible to have many CLOG page
 * requests in flight at one time which could lead to disk access for CLOG
 * page if the required page is not found in memory.  Testing revealed that we
 * can get the best performance by having 128 CLOG buffers, more than that it
 * doesn't improve performance.
```

<-- clog.c @ 9.6

CLogControlLock

- From *PostgreSQL hackers' mailing list*
  - In my investigation, I *(Amit Kapila)* found that the contention is mainly due to two reasons,
  - one is that while writing the transaction status in CLOG (TransactionIdSetPageStatus()), it acquires EXCLUSIVE CLogControlLock which contends with every other transaction which tries to access the CLOG for checking transaction status and to **<u>reduce it already</u>** a patch [1] is proposed by Simon;
  - Second contention is due to the reason that when the CLOG page is not found in CLOG buffers, it needs to acquire CLogControlLock in Exclusive mode which again contends with shared lockers which tries to access the transaction status.

*

http://www.postgresql-archive.org/Speed-up-Clog-Access-by-increasing-CLOG-buffers-td5864147.html

Simple is best, if possible

- The reason why buffer replacement is necessary?
  - CLOG buffer capacity is not enough.

    128 buffers --> 128 x 32K (= 4M) TRXs

Starting point

- If memory is abundant
  - Status for **all transactions** can be stored in the shared memory, enabling **direct mapping** of a TXID and corresponding status bits

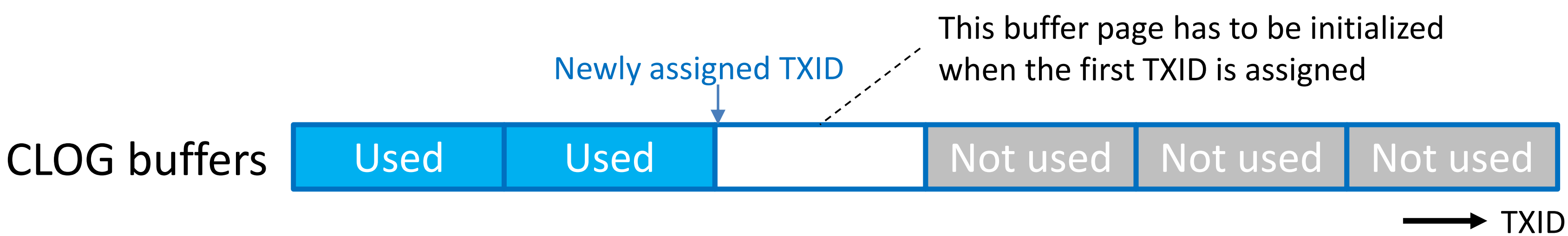
    --> No buffer replacement, no linear search

Array of CLOG buffers

Memory location of status bits for each transaction is corresponds to its TXID

Benefits of direct mapping

- Elimination of ‘CLOG page not found’ events
  - CLogControlLock requests due to page-not-found events are also eliminated
  - Buffer initialization occurs, which accompanies the progress of the transaction ID



- Decrease in the access time for the status bit
  - resulting in the decrease in the CLogControlLock holding time

How many TXs?

- Factors related to this matter
  - TXID is 32-bits length

    --> Max. 4G TXs
  - 'autovacuum_freeze_max_age' due to the XID wraparound problem

    --> It does not exceed 2G TXs

How much shared memory?

- Transaction status (2bit/TX)

$$2G_{\text{TX}} \times \frac{2}{8}_{\text{Byte/TX}} = 0.5G_{\text{Bytes}}$$

- ‘group_lsn[]’

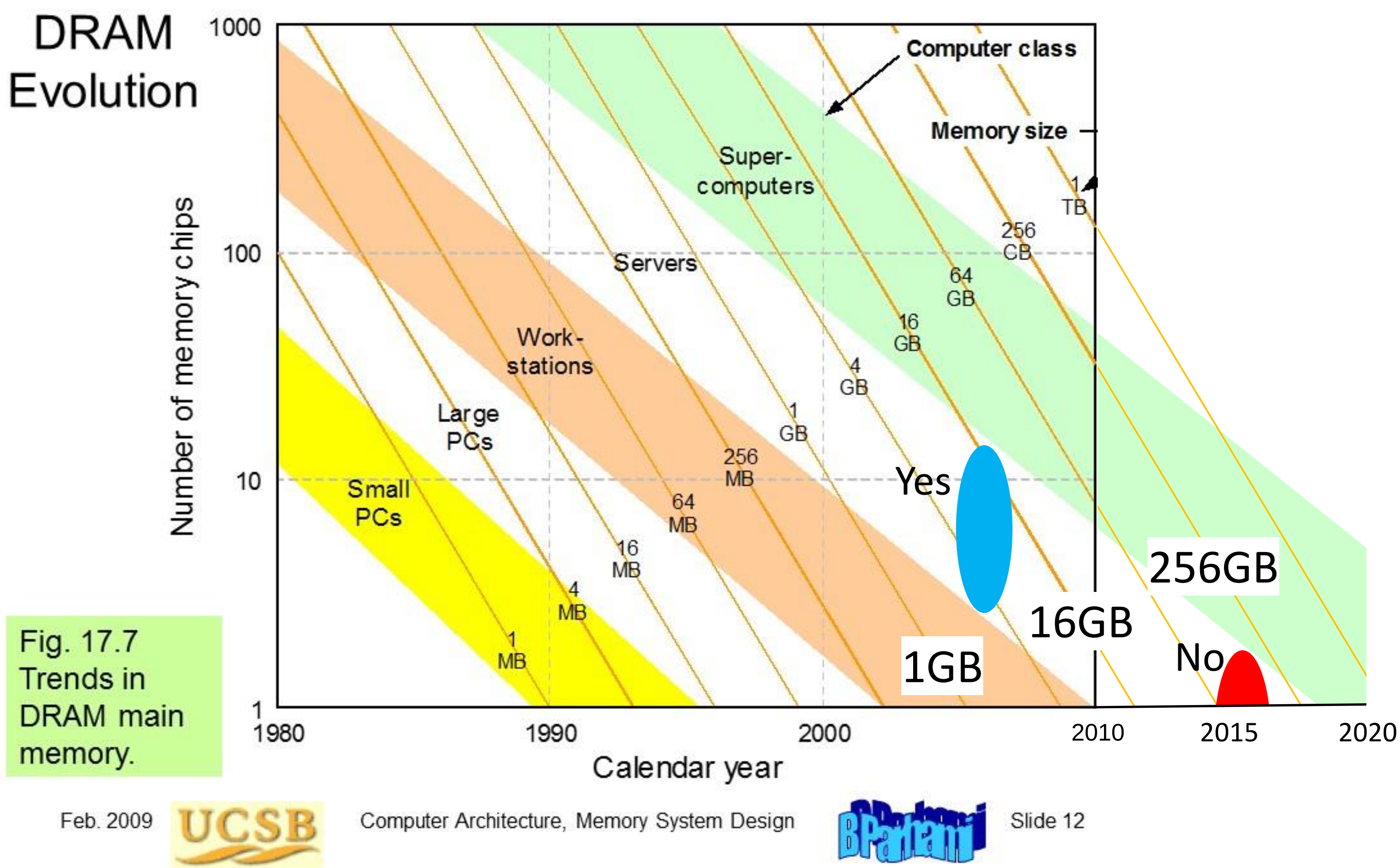
# of CLOG pages X CLOG_LSNS_PER_PAGE X sizeof(XLogRecPtr) = $0.5G_{\text{Bytes}}$

See SimpleLruShmemSize() @ clog.c

Total $1G_{\text{Bytes}}$

Moore save the Amdahl

- Is 1G-Bytes of shared memory too much?



Linearly extend the trends shown in https://www.slideshare.net/Flashdomain/computer-architecture-part-5

Implementation

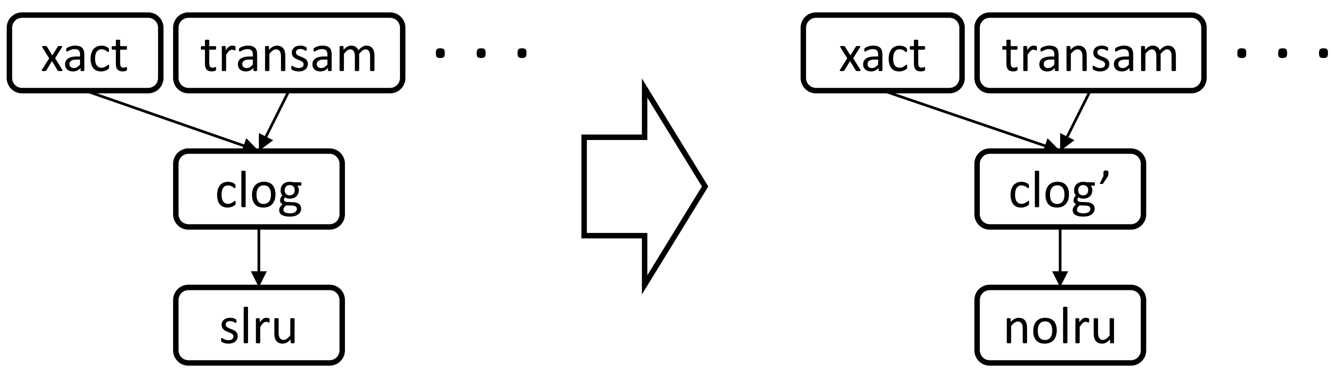
https://github.com/meistervonperf/postgresql-NoCLogLru

- Changes in the source code
  - Modified

    src/backend/access/transam/clog.c ~40 lines
    src/backend/access/transam/Makefile 1 line
  - Added

    src/backend/access/transam/nolru.c ~1.2K lines
    src/include/access/nolru.h ~200 lines

Based on slru.[c|h]



Original Changed

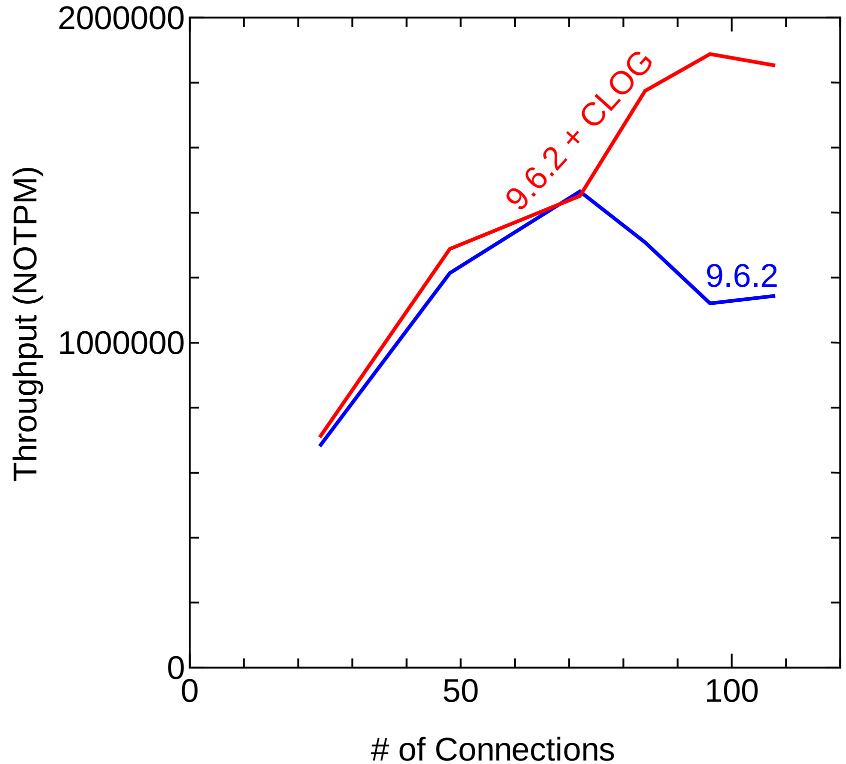
Experimental setup

- Hardware
  - DB server
    CPU: E7-8890v4@2.20GHz x 4 (24 cores x 4 = 96 cores)
    Memory: 1TB
    FC Storage
      RAID10: 15Krpm/600GB x 16 for data
      RAID10: 15Krpm/600GB x 32 for WAL
  - Client
    CPU: E5-2699v3@2.3GHz x 2 (18 cores x 2)
    Memory: 768GB
  - Network
    GB ether x 4

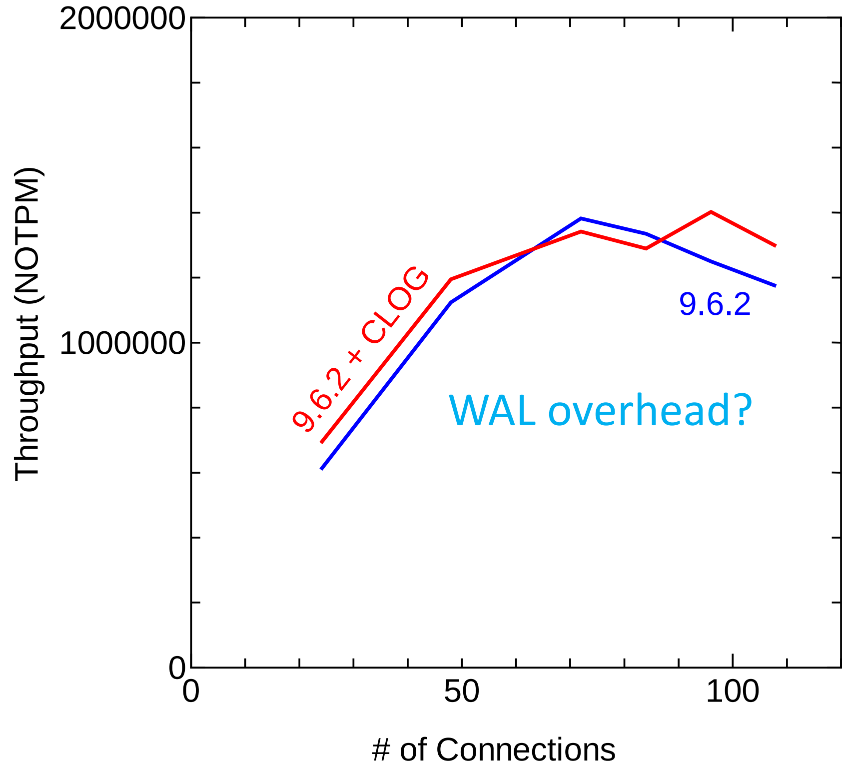
- Software, workload, etc.
  - Benchmark : DBT-2
  - DBMS : PostgreSQL 9.6.2
  - OS : Linux 3.10.0 (CentOS 7.3)

Performance evaluation

Benchmark : DBT-2



Unlogged table



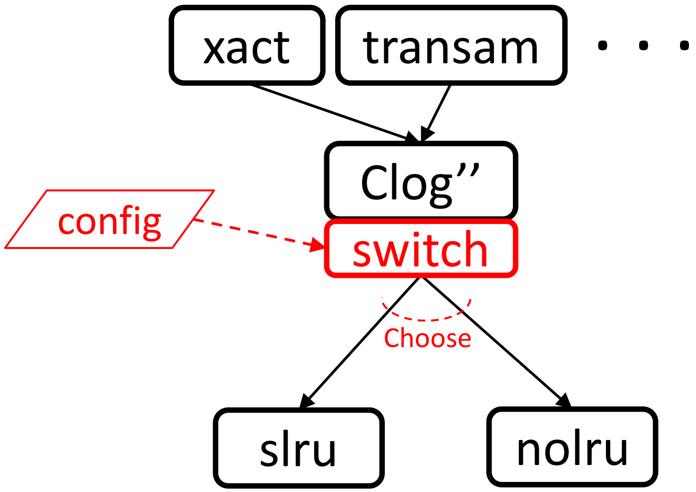
Logged table

Consideration for small memory

- Not all machines are equipped with large memory
- Current LRU mechanism is suitable for small memory machine

⇨ It is necessary to be able to choose proper clog mechanism from that using LRU replacement and that employing direct mapping

(Not implemented yet)

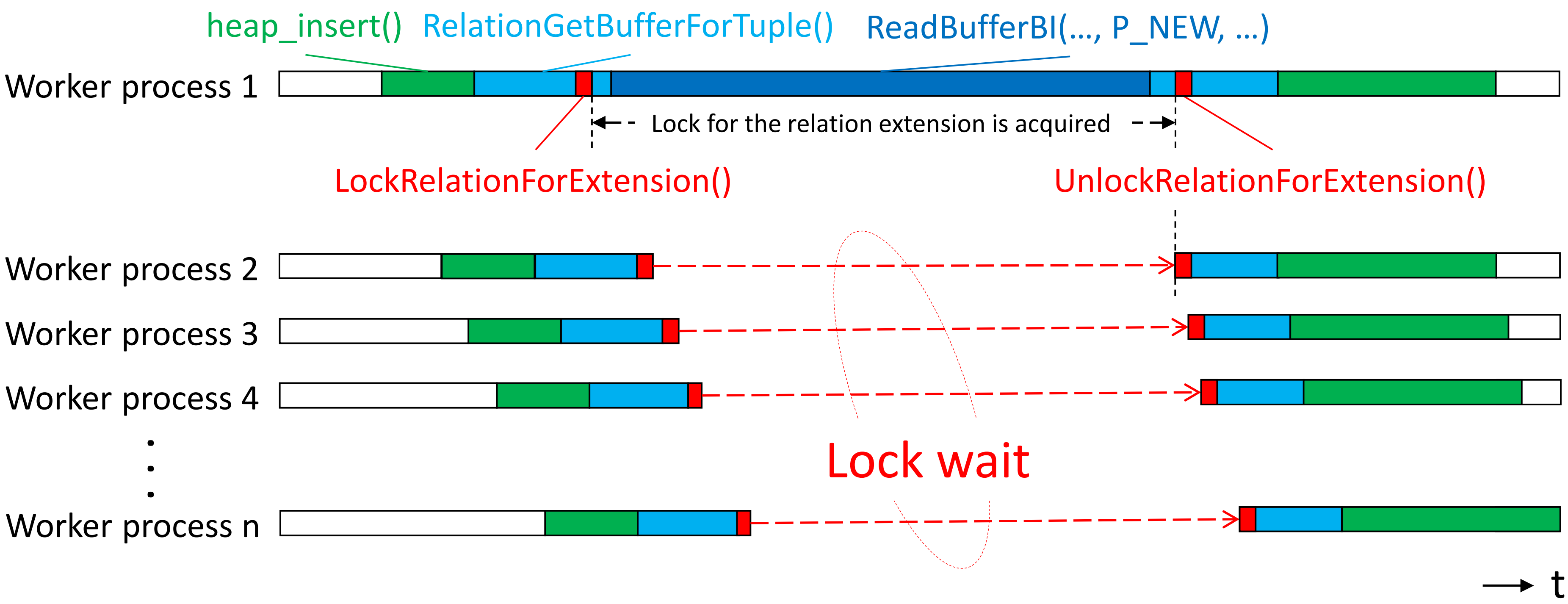


Agenda

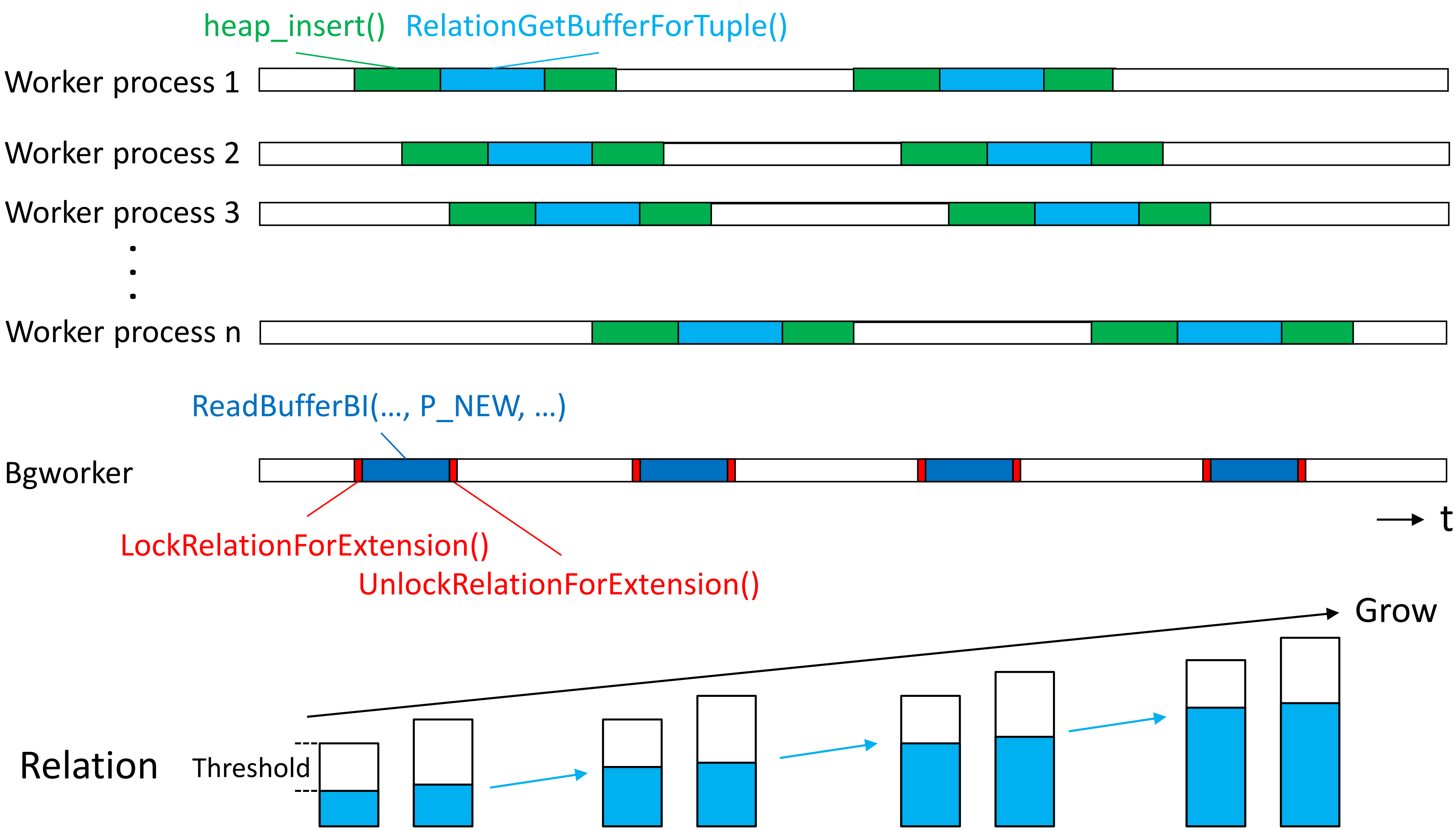
- Introduction
  - Start with trend in computer architecture
- Simple is best
  - CLogControlLock
- Prepare in advance
  - Table extension
- Achievements to date (Review)
  - Countermeasures for CPU scalability bottlenecks
- Concluding remarks

It’s too late

- An example : extension of a relation

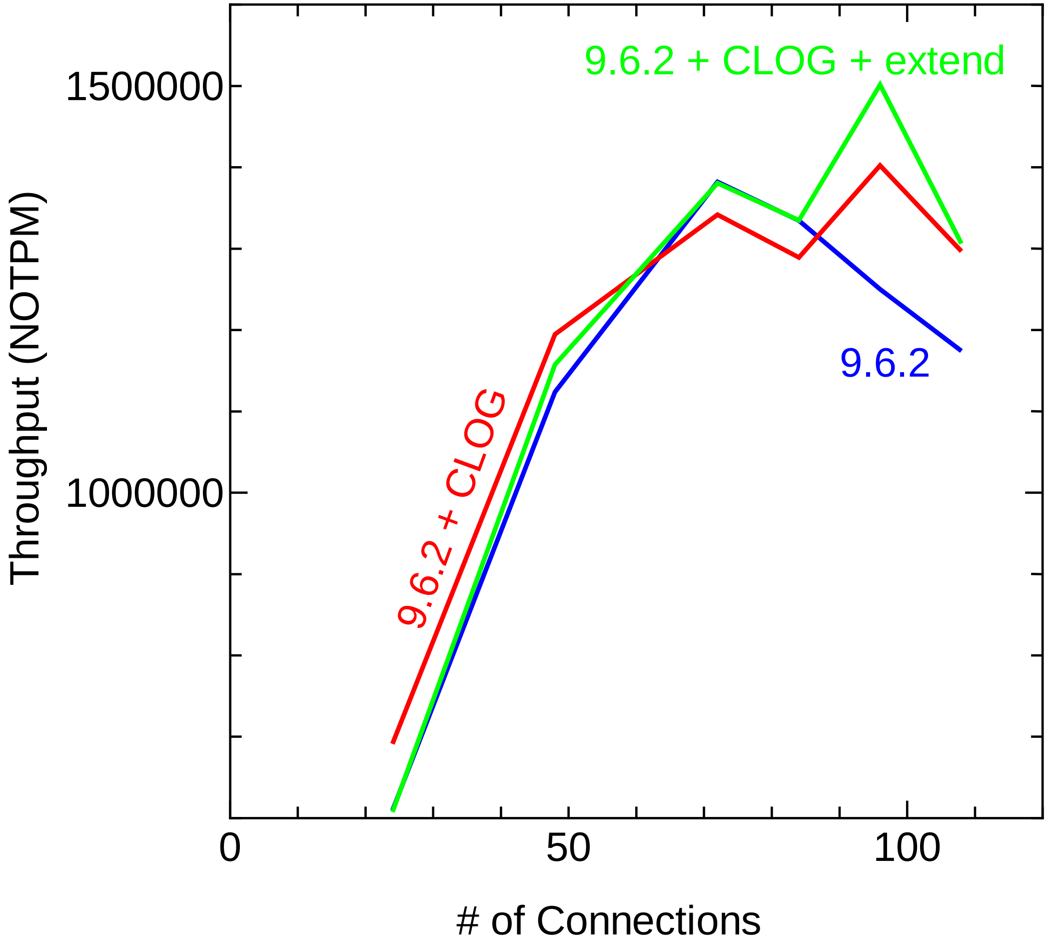


Prepare in advance



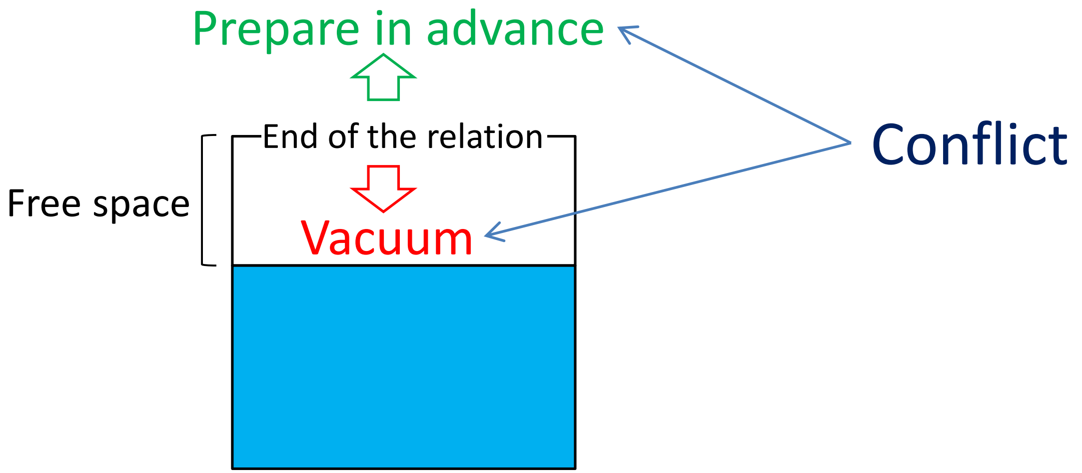
Performance evaluation

Benchmark : DBT-2
Using logged table



Unresolved issue

- Conflict with existing mechanism (vacuum)
  - Vacuum tries to truncate a relation when there is free space in it, which is carried out by lazy_truncate_heap()
  - 'Prepare in advance' strategy tries to make a certain amount of space at the end of the relation



Similar structure

```
TransactionId
GetNewTransactionId()
{
    . . .
    LWLockAcquire(XidGenLock, LW_EXCLUSIVE);
    . . .

    /
     * If we are allocating the first XID of a new page of the commit log,
     * zero out that commit-log page before returning. We must do this while
     * holding XidGenLock, else another xact could acquire and commit a later
     * XID before we zero the page.  Fortunately, a page of the commit log
     * holds 32K or more transactions, so we don't have to do this very often.
     *
     * Extend pg_subtrans and pg_commit_ts too.
     */
    ExtendCLOG(xid);
    ExtendCommitTs(xid);
    ExtendSUBTRANS(xid);
    . . .

    LWLockRelease(XidGenLock);
    . . .
}
```

As the TXID progresses page initialization is performed periodically with holding XidGenLock, which makes other processes that request a new TXID wait for the lock.

Agenda

Introduction

- Start with trend in computer architecture

Simple is best

- CLogControlLock

Prepare in advance

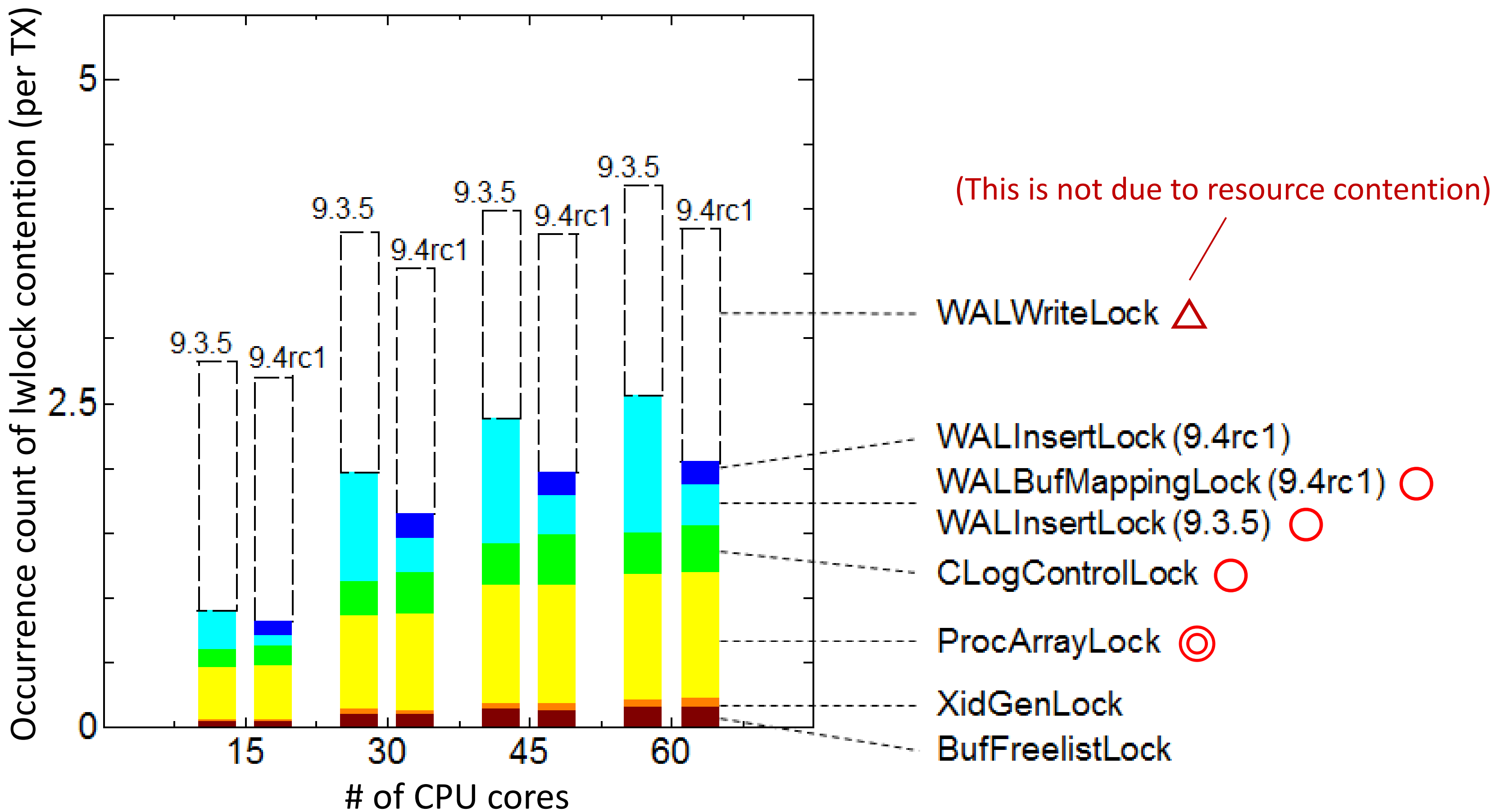
- Table extension

Achievements to date (Review)

- Countermeasures for CPU scalability bottlenecks

Concluding remarks

A measurement of lock contention



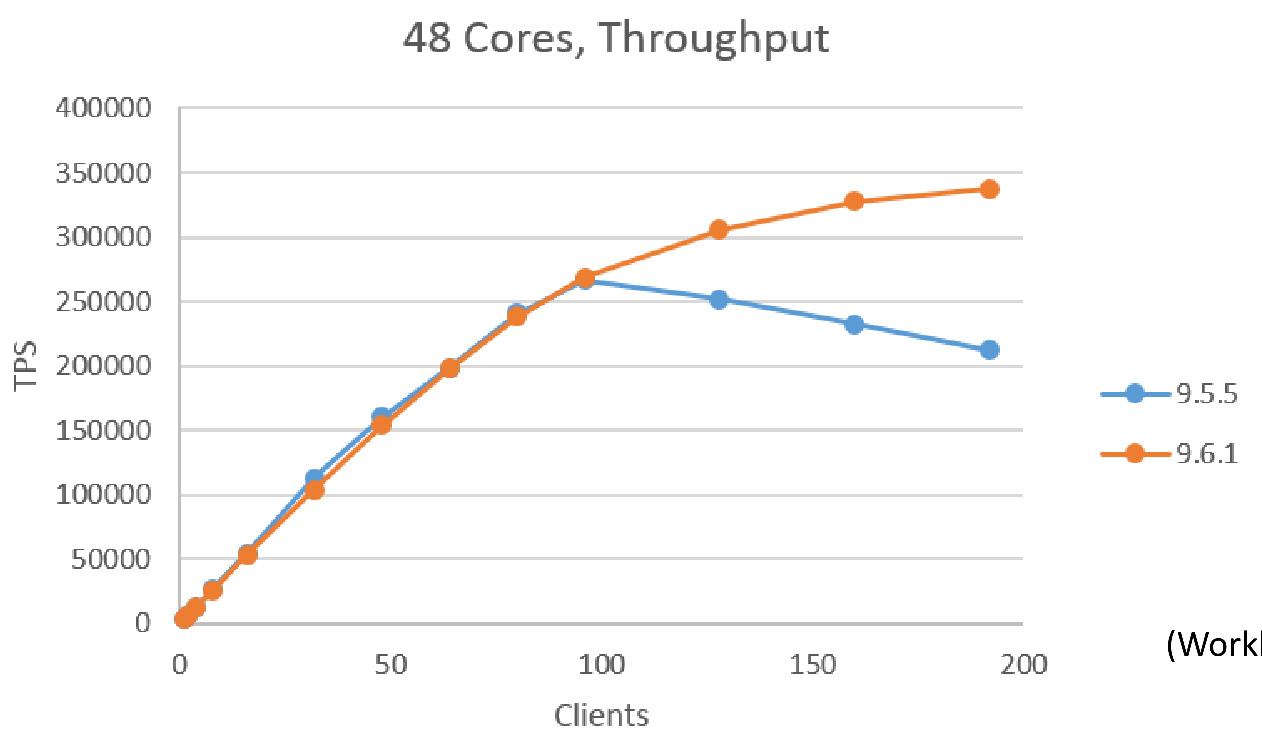
Report of 2014 PGECons (PostgreSQL Enterprise Consortium) WG1 Activity, PGECons JAPAN, https://www.pgecons.org/downloads/89

Achievements to date

- 9.6
  - Reduce contention for the ProcArrayLock (Amit Kapila, Robert Haas) ProcArrayLock
  - Partition the shared hash table freelist to reduce contention on multi-CPU-socket servers (Aleksander Alekseev) SpinLock for hash freeList
  - Extend relations multiple blocks at a time when there is contention for the relation's extension lock (Dilip Kumar) LockRelationForExtension()
  - Increase the number of clog buffers for better scalability (Amit Kapila, Andres Freund) CLogControlLock
- 9.5
  - Increase the number of buffer mapping partitions (Amit Kapila, Andres Freund, Robert Haas) LWLocks for BUFFER_MAPPING
  - Improve lock scalability (Andres Freund) LWLock mechanism
- 9.4
  - Allow multiple backends to insert into WAL buffers concurrently (Heikki Linnakangas) WALInsertLock
- 9.2
  - Allow uncontended locks to be managed using a new fast-path lock mechanism (Robert Haas) LockRelationOid()
  - Make the number of CLOG buffers scale based on shared_buffers (Robert Haas, Simon Riggs, Tom Lane) CLogControlLock

A report on the ProcArrayLock tuning

- It contributes to performance improvement in areas with a large number of clients



(Workload : pgbench)

Report of 2016 PGECons (PostgreSQL Enterprise Consortium) WG1 Activity, PGECons JAPAN, https://pgecons-sec-tech.github.io/tech-report/html_wg1_2016/wg1_2016.html

Agenda

- Introduction
  - Start with trend in computer architecture
- Simple is best
  - CLogControlLock
- Prepare in advance
  - Table extension
- Achievements to date (Review)
  - Countermeasures for CPU scalability bottlenecks
- Concluding remarks

Concluding remarks

- Exploiting parallelism becomes more and more important, due to spread use of many-core processors.
  - The priority of countermeasures for bottlenecks should be determined with considering the performance impact.
  - CLOG is a top priority target, as well as WAL insertion mechanism.
- It becomes feasible to place CLOG of all transactions in main memory.
  - It contributes the decrease in the contention on CLogControlLock, resulting in CPU scalability improvement.
- 'Prepare in advance' strategy is a possible countermeasure for bottlenecks arising in extending a relation, CLOG, etc.
  - It is necessary to adapt existing systems so that new mechanism can work effectively.
  - It is also necessary to study about its effect further.

Thank you for listening

Any questions?