Подключаемые движки хранения в PostgreSQL

Александр Коротков

Postgres Professional

2018

Дисклеймер

В разработке подключаемых движков хранения заинтересованы различные стороны. Не вся информация, представленная в данном докладе, согласована со всеми заинтересованными сторонами.

Pluggable storage engines? Зачем?

- Пока у постгреса один движок хранения, и бывает он подвергается критике.
- Постгрес изначально разработан в духе расширяемости. Отсутствие подключаемых движков хранения – досадное упущение.
- У многих компаний разрабатываются свои движки: Postgres Pro (in-memory OLTP), EnterpriseDB (zheap), 2ndQuadrant (columnar on-disk), Fujitsu (columnar in-memory), Citus (columnar on-disk).

Heap with undo log by EnterpriseDB

http://rhaas.blogspot.ru/2018/01/do-or-undo-there-is-no-vacuum.html

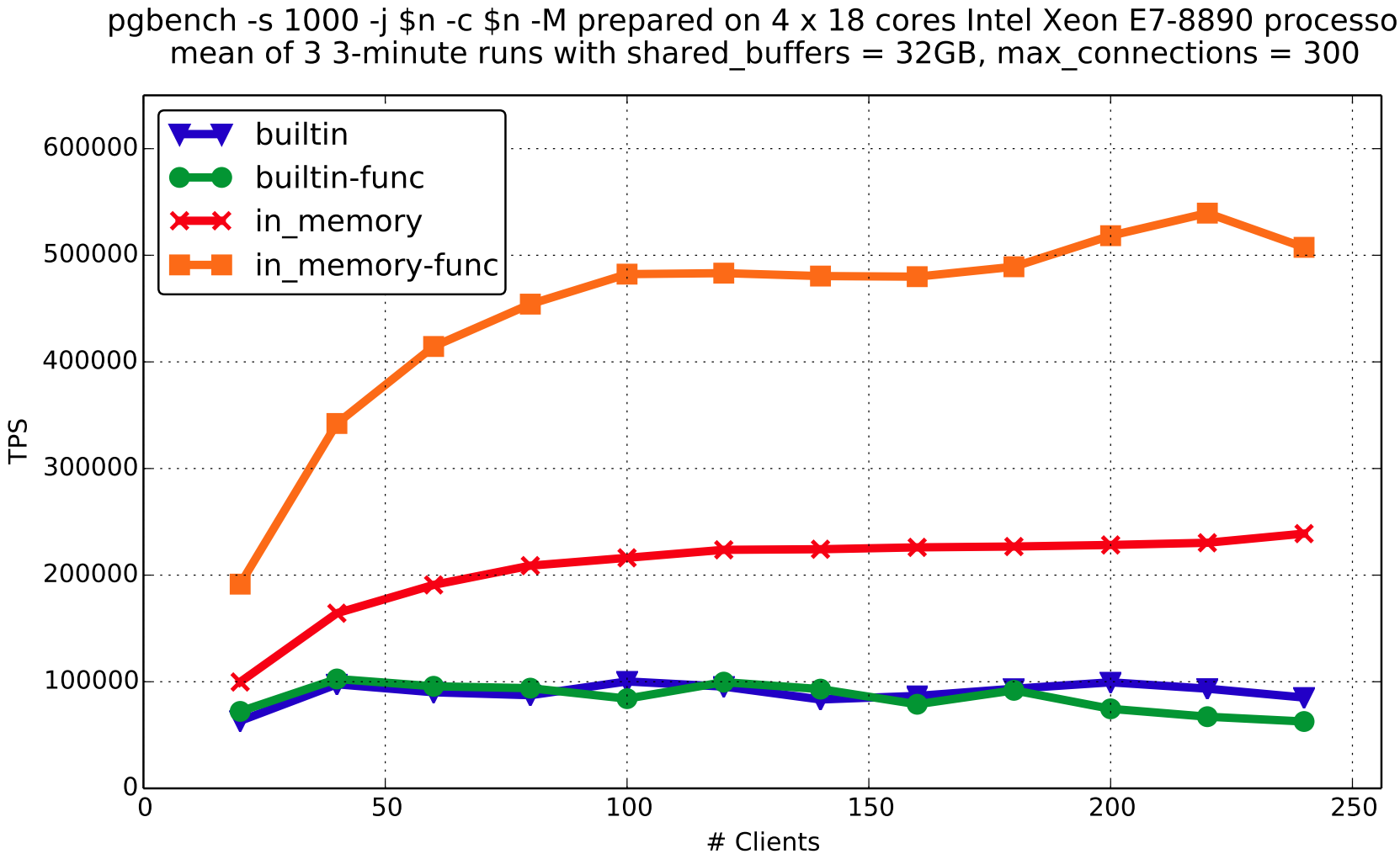
DO or UNDO - There is no VACUUM

Author: Robert Haas
1/30/2018

What if PostgreSQL didn't need VACUUM at all? This seems hard to imagine. After all, PostgreSQL uses multi-version concurrency control (MVCC), and if you create multiple versions of rows, you have to eventually get rid of the row versions somehow. In PostgreSQL, VACUUM is in charge of making sure that happens, and the autovacuum process is in charge of making sure that happens soon enough. Yet, other schemes are possible, as shown by the fact that not all relational databases handle MVCC in the same way, and there are reasons to believe that PostgreSQL could benefit significantly from adopting a new approach. In fact, many of my colleagues at EnterpriseDB are busy implementing a new approach, and today I'd like to tell you a little bit about what we're doing and why we're doing it.

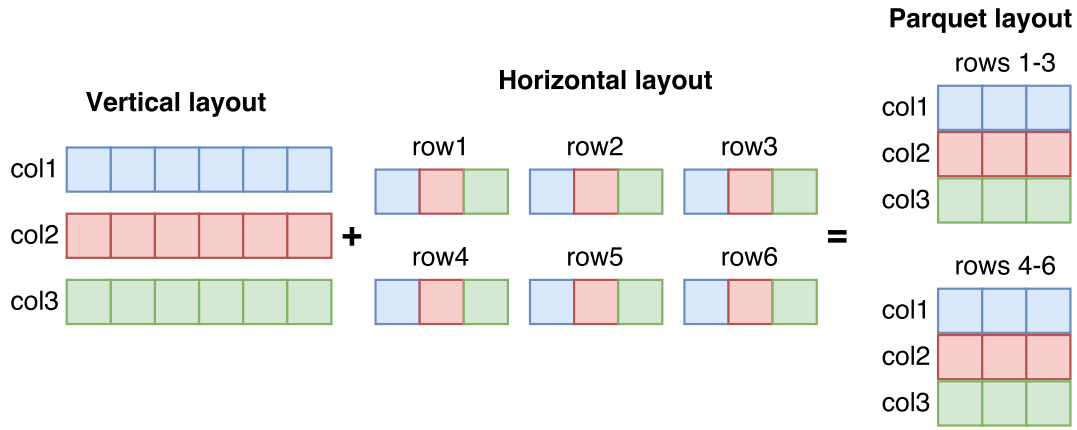
In-memory OLTP FDW by Postgres Professional

https://goo.gl/nD6kdC



vops – parquet layout by Postgres Professional

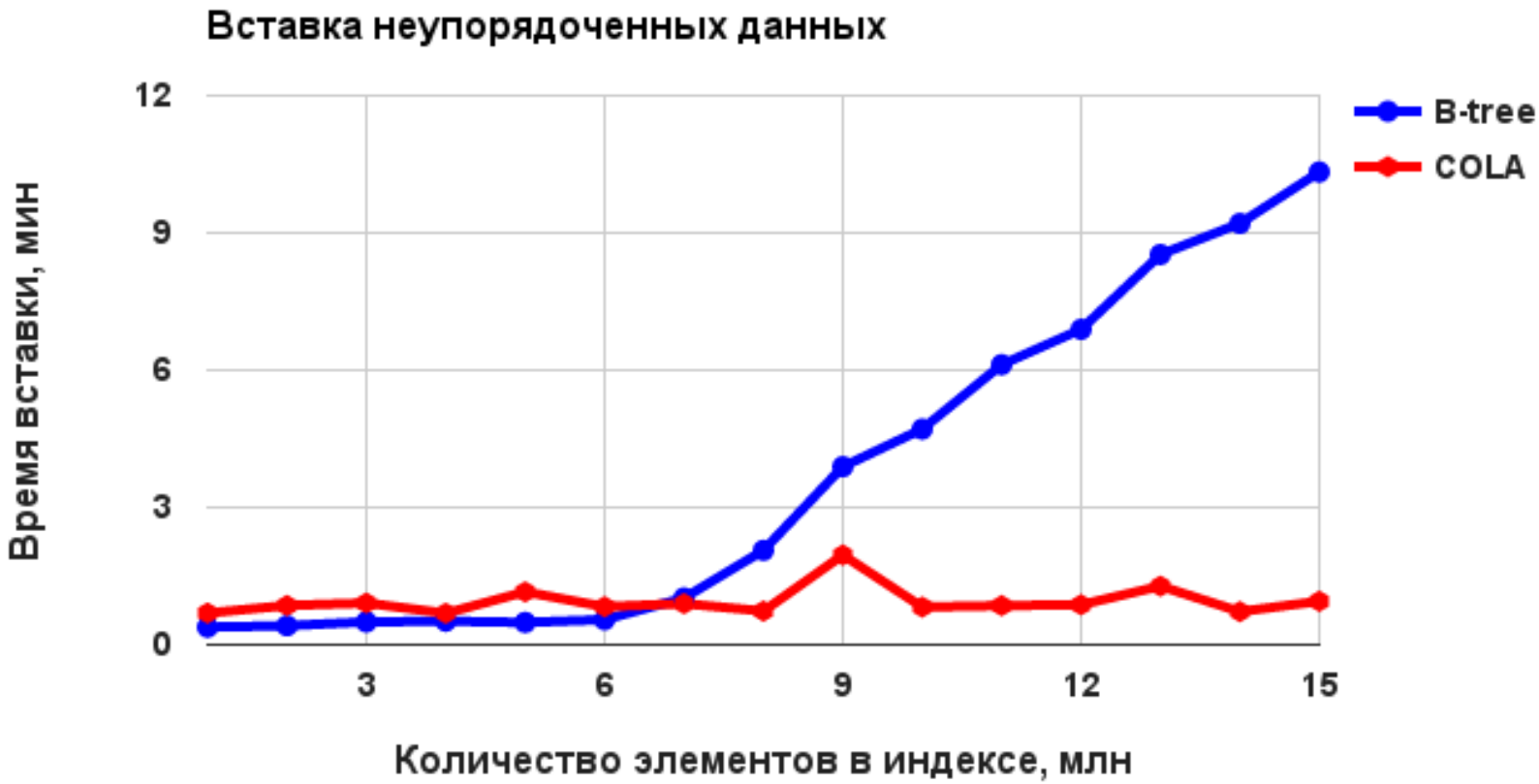
https://github.com/postgrespro/vops



Query	Seq. exec., msec	Parallel exec. (msec)
Original Q1 for lineitem	38028	10997
Original Q1 for lineitem_projection	33872	9656
Vectorized Q1 for vops_lineitem	3372	951
Mixed Q1 for vops_lineitem_projection	1490	396
Original Q6 for lineitem	16796	4110
Original Q6 for lineitem_projection	4279	1171
Vectorized Q6 for vops_lineitem	875	284

COLA index (LSM-tree) by Postgres Professional

В перспективе нужен не только индекс, но весь движок.



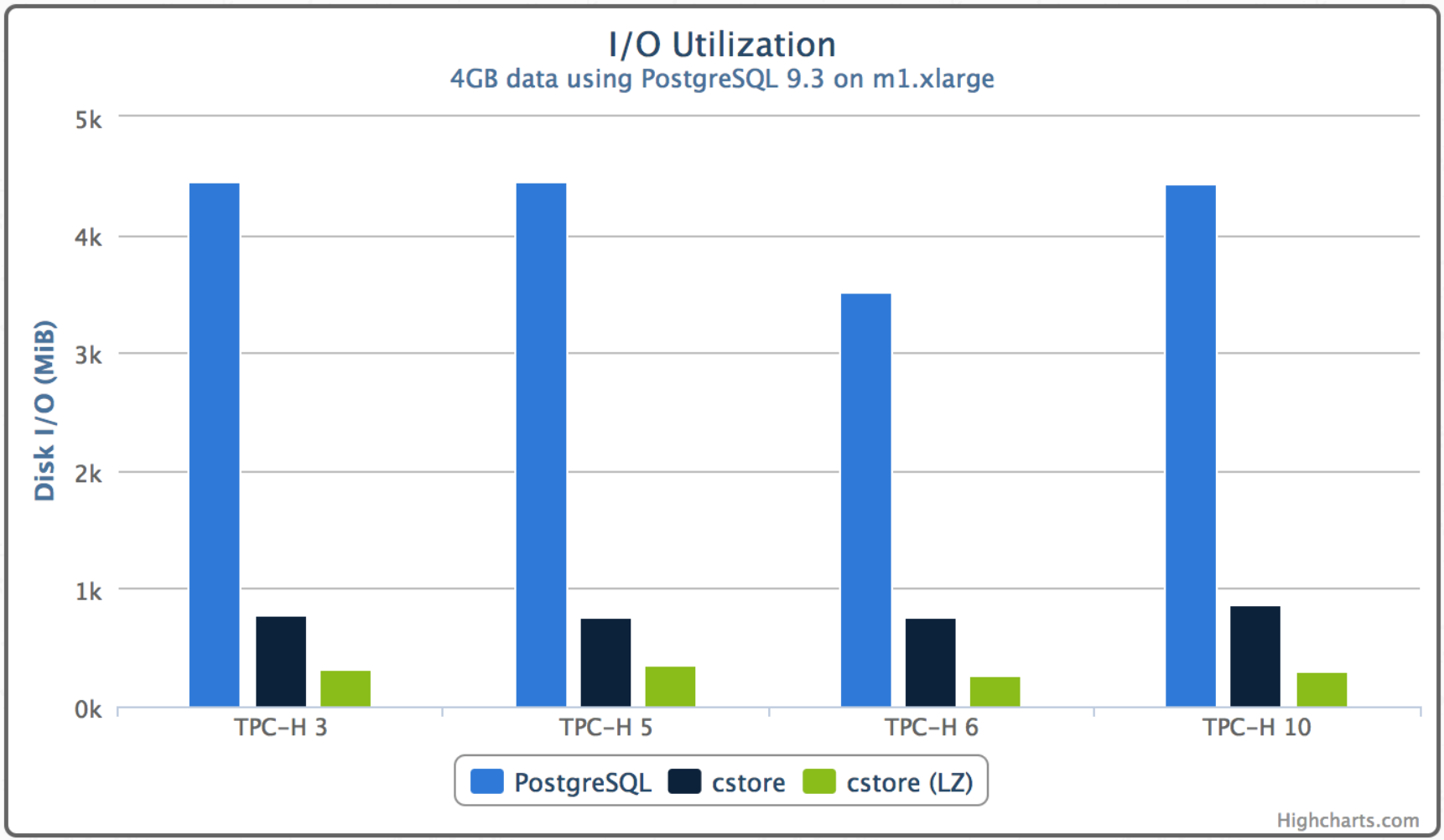
In-memory columnar enigne by Fujitsu

https://www.pgcon.org/2016/schedule/events/919.en.html

	PostgreSQL	VCI	VCI	VCI Performance	VCI Performance	RDBMS1	RDBMS1	RDBMS2	RDBMS3	RDBDMS3 40CPU	RDBDMS3 1CPU	RDBMS3 40CPU
	VCI off	1CPU	40CPU	1CPU	40CPU	Column	Row		1CPU Row	40CPU Row	1CPU In memory option	40CPU In memory option
	Execution time (ms)	Execution time (ms)	Execution time (ms)	Execution time (ms)	Execution time (ms)	Execution time (ms)	Execution time (ms)	Execution time (ms)	Execution time (ms)	Execution time (ms)	Execution time (ms)	Execution time (ms)
Q1	91,841	26,090	1,004	25,942	975	4,312	2,227	532	38,935	1,618	32,307	1,119
Q2	4,894	4,784	4,449	4,963	2,060	1,387	4,606	1,601	531	685	302	1,006
Q3	28,999	16,996	14,690	16,805	2,701	8,716	5,945	2,836	1,678	1,473	908	954
Q4	5,000	5,373	3,179	5,097	843	1,545	2,784	2,484	15,701	1,190	8,825	628
Q5	13,709	8,431	6,790	9,006	1,068	1,617	4,694	1,420	21,824	1,798	10,199	2,213
Q6	9,015	7,516	498	7,691	534	284	416	154	8,796	692	453	51
Q7	14,502	13,110	14,029	13,958	1,674	1,342	19,567	420	18,568	1,848	6,125	1,415
Q8	5,253	5,022	4,604	4,884	605	1,887	19,233	1,504	17,348	2,024	3,558	1,897
Q9	69,843	69,138	67,276	66,411	6,387	150,000	535,358	1,214	33,695	4,904	18,217	2,587
Q10	11,064	11,376	9,152	11,430	3,522	743	4,584	3,812	4,718	1,526	887	1,371
Q11	1,315	1,171	1,118	1,298	1,319	616	306	763	2,689	724	763	607
Q12	12,610	17,475	2,329	17,468	718	571	34,863	720	13,815	1,134	6,188	651
Q13	60,433	59,243	58,342	60,309	60,454	1,899	11,358	6,763	28,415	1,261	21,574	898
Q14	3,993	8,470	2,347	8,546	1,517	1,787	750	191	15,046	1,013	1,768	495
Q16	20,252	19,796	19,198	20,202	19,332	1,037	7,357	605	4,153	1,326	2,805	1,371
Q17	1,877	1,680	1,408	1,631	209	276	1,640	9,420	561	261	2,128	503
Q18	43,769	42,302	42,602	44,541	43,099	3,647	21,768	1,840	16,720	1,587	13,862	2,227
Q19	1,486	1,080	395	1,105	99	214	26,308	297	1,350	100	1,074	309
Q20	2,013	1,859	1,605	1,977	303	179	10,797	564	2,926	20,100	2,299	2,441
Q21	24,863	24,630	24,228	25,495	2,323	13,794	34,946	1,557	167,817	8,360	167,417	6,998
Q22	3,270	2,503	1,023	2,630	331	569	6,872	305	2,189	524	3,368	775

cstore_fdw by Citus Data

https://citusdata.github.io/cstore_fdw/



Columnar storage by 2ndQuadrant

https://blog.2ndquadrant.com/column-store-plans/

Column Store Plans

SHARE

April 25, 2016 by alvherre

Over at pgsql-general, Bráulio Bhavamitra asks:

I wonder if there is any plans to move postgresql entirely to a columnar store (or at least make it an option), maybe for version 10?

This is a pretty interesting question. Completely replacing the current row-based store wouldn't be a good idea: it has served us extremely well and I'm pretty sure that replacing it entirely with a columnar store would be disastrous performance-wise for OLTP use cases.

FOSDEM PostgreSQL Developer Meeting 2018

Общая идея

У движков хранения **разные**:

- Способы сканировать и изменять таблицы,
- реализации MVCC;

(иначе это уже не будут разные движки хранения)

общие:

- транзакции и снапшоты,
- WAL.

(иначе это уже не будет одна СУБД)

С чего начать?

- Extend FDW
  - **Pro:** сразу довольно много свободы.
  - **Cons:** по началу подключаемые движки хранения будут сильно “обделены” функциональностью.
- Pluggable heap
  - **Pro:** подключаемые движки хранения сразу будут “first class citizen”.
  - **Cons:** свобода будет доставаться небольшими порциями, достигаемыми рефакторингом.

План реализации

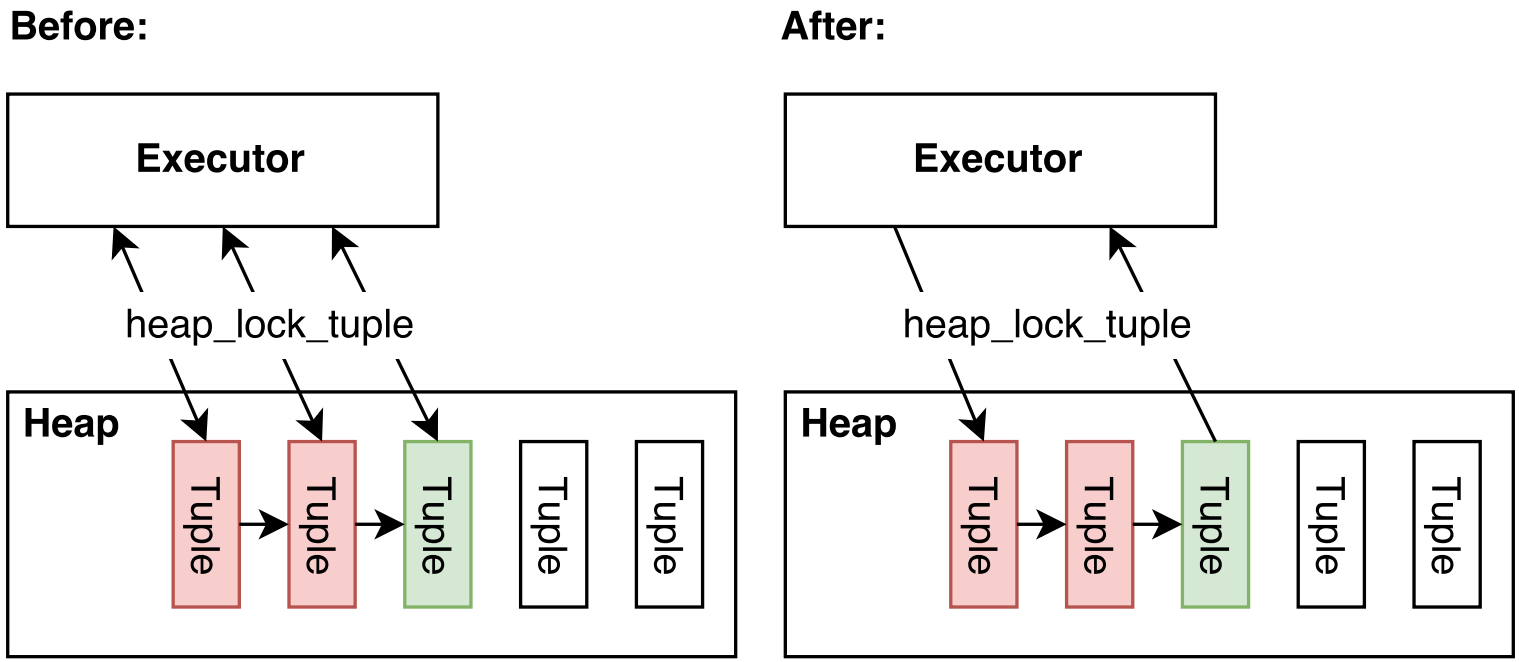
- Этап 1: tid row locator (zheap).
- Этап 2: non-tid row locator (index-organized table).
- Этап 3: no row locator (columnar).

WAL

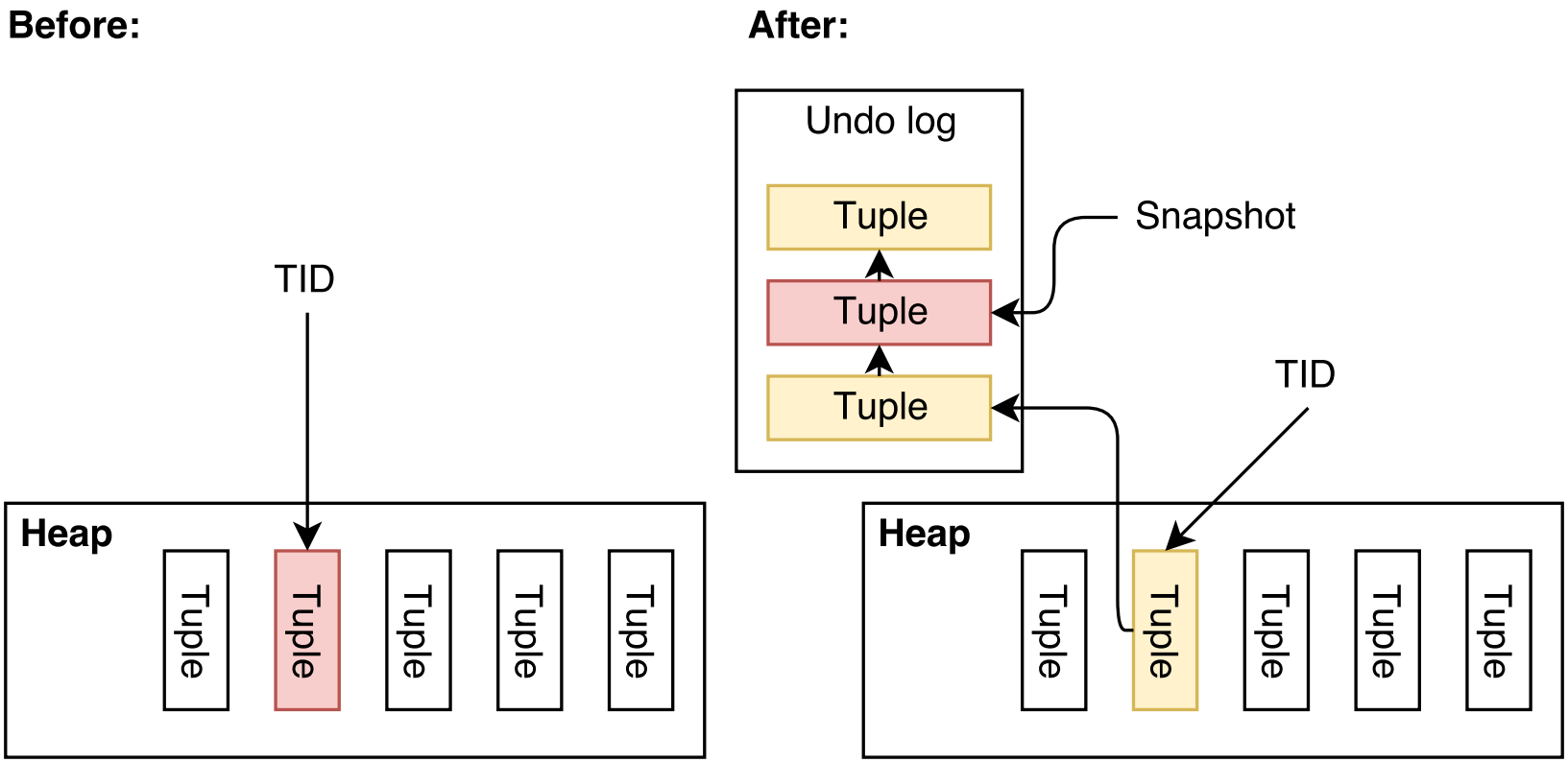
Варианты решения:

- Встроенные движки, подключаемые через API могут иметь свои redo функции,
- Generic WAL,
- 2 phase recovery: восстанавливаем встроенные движки, а потом движки-расширения.

Tuple lock


Before:
Executor
heap_lock_tuple
Heap
Tuple
Tuple
Tuple
Tuple
Tuple
After:
Executor
heap_lock_tuple
Heap
Tuple
Tuple
Tuple
Tuple
Tuple

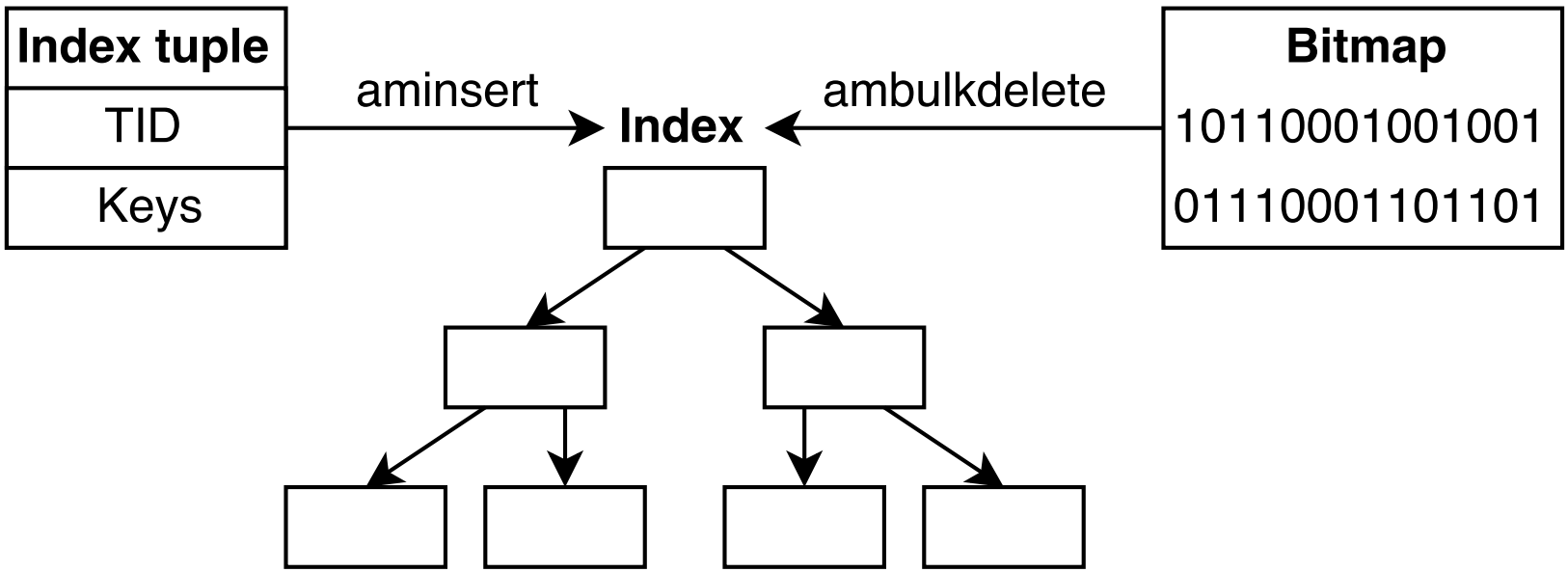
Tuple identifier vs row identifier



Взаимодействие с index access methods

- Не хочется писать отдельные index access methods для каждого движка хранения.
- Поэтому нужно совершенствовать index access methods API, чтобы он подходил для разных движков.

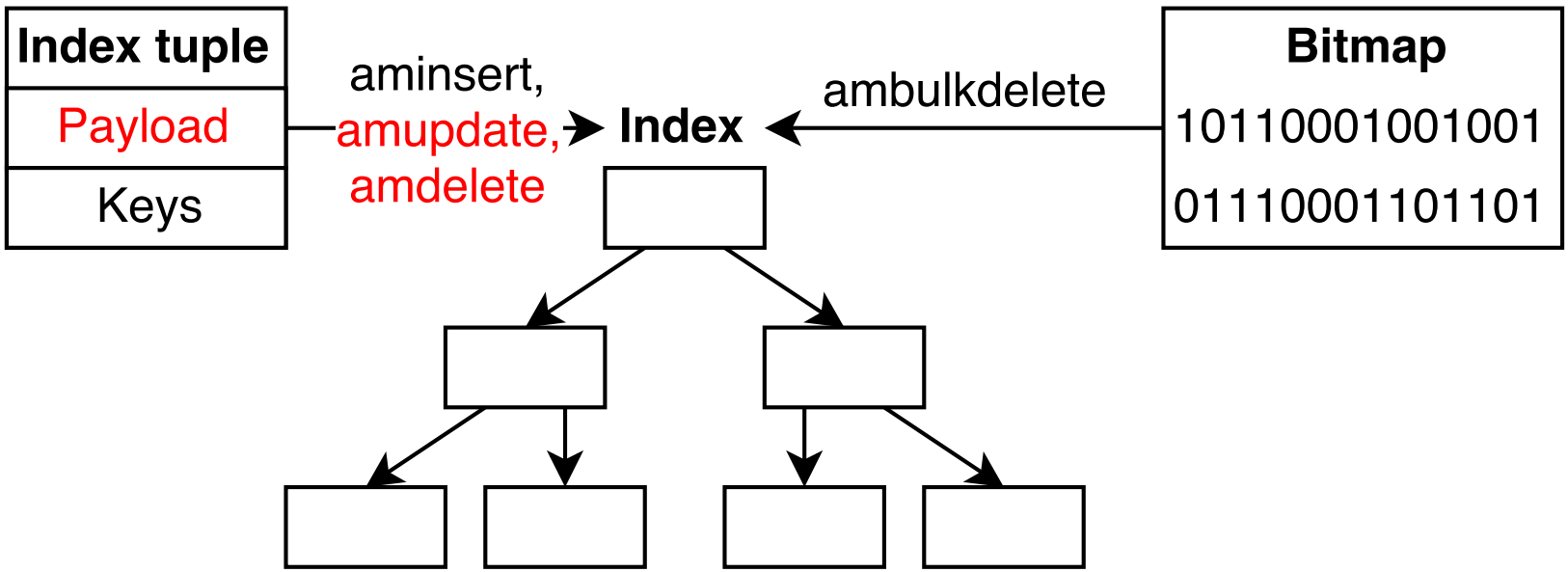
Вставка/удаление в/из индекса



Совершенствование index access methods API

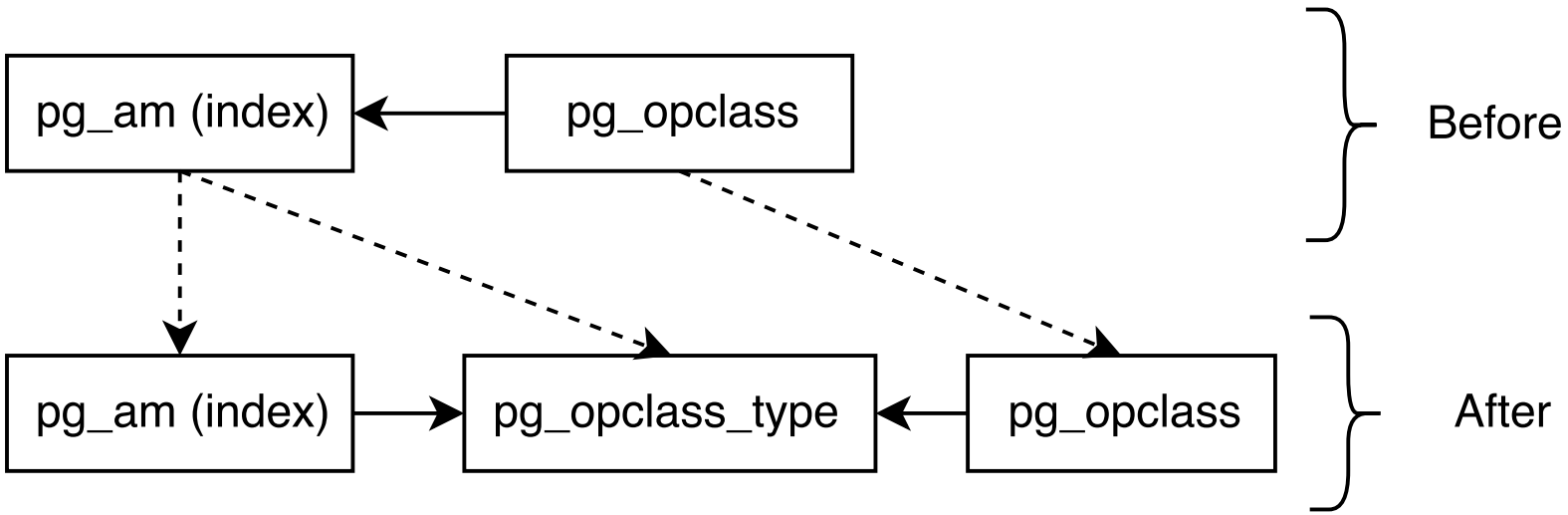
- Возможность удаления одного tuple.
- Хранение произвольной “полезной нагрузки” tuple вместо TID.
- Обновление “полезной нагрузки” tuple.

Усовершенствованное index access methods API

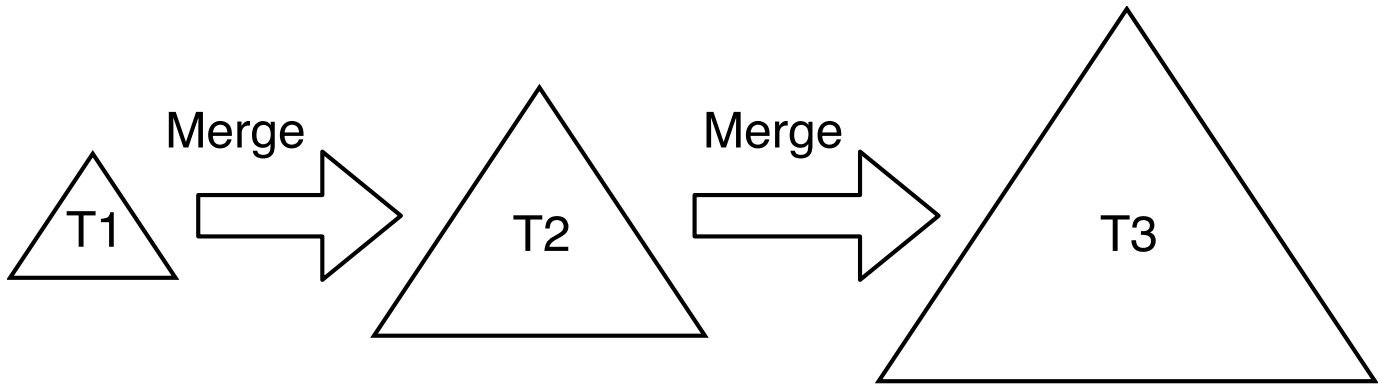


Переиспользуемые opclass’ы?

Новый access method’ы должны иметь возможность переиспользовать существующие opclass’ы. Например, LSM может переиспользовать B-tree opclass’ы.



LSM engine



- Index-organized table, где index – LSM,
- Для полноценной и корректной реализации нужна поддержка “слепого” UPDATE (и UPSERT) на уровне SQL.

Набор патчей для подключаемых движков хранения

- См. mailing list thread [1]
- Последняя ревизия включает в себя 12 патчей.
- Разработчики: Alvaro Herrera, Haribabu Kommi, Alexander Korotkov.
- Набор патчей упорядочен следующим образом: вначале введение API, потом рефакторинг. Нужно наоборот!

[1] https://www.postgresql.org/message-id/20160812231527.GA690404@alvherre.pgsql

Спасибо за внимание!